REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE

3. REPORT TYPE AND DATES COVERED
FINAL 01 JUL 94 TO 31 DEC 95

4. TITLE AND SUBTITLE
DISTRIBUTED OPTIMIZATION FOR CUSTOMIZED AIRCRAFT FLEET SCHEDULING

5. FUNDING NUMBERS
F49620-94-1-0411
3484/BS
61103D

6. AUTHOR(S)
KENDALL E. NYGARD

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NORTH DAKOTA STATE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND IPERATIONS RESEARACH
FARGO, ND 58105-5164

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR/NM
1110 DUNCAN AVE , SUITE B115
BOLLING AFB, DC 20332-8080

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE:
DISTRIBUTION UNLIMITIED

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)
A COMPLETE SOFTWARE SYSTEM FOR MILITARY PASSENGER AIRCRAFT SCHEDULING WAS DESIGNED AND BUILT.

19960726 084

14. SUBJECT TERMS

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

Distributed Optimization for Customized Aircraft Fleet Scheduling

Interim Technical Report

July 31, 1995

Submitted to:

Air Force Office of Scientific Research

by

North Dakota State University

Grant F49620-94-1-0411

Kendall E. Nygard
Department of Computer Science and Operations Research
North Dakota State University
Fargo, ND 58105-5164

TABLE OF CONTENTS

1. INTRODUCTION

DAKOTA is a software environment for aircraft scheduling. Important features of the system include an extended set partitioning algorithm for optimizing schedules, a geographic-based display subsystem, schedule editing tools, context-sensitive hyperhelp, and report writing facilities. The goal is to provide a system that can help users quickly produce aircraft schedules that efficiently utilize the available fleet. The manual schedule editing procedures and the optimizer work seamlessly, making the best of both approaches available to the scheduler. Designed from the perspective of the end user, DAKOTA is a comprehensive flight scheduling system that is easy to use, and easy to master.

Designing and implementing high-quality graphical human-computer interfaces is a challenging process. In this report, some of the user interface design choices are presented and analyzed. The fundamental problem addressed is the need to arrange customized air travel military personnel (clients) who submit travel requests composed of one or more origin-destination pairs called request legs. Along with the request, there are constraints, expressed in terms of the time when the client wishes to arrive at the destination, the time when the client wants to leave from the origin, and the type of aircraft on which he/she wishes. In addition, there are constraints from the operational point of view, which involve aircraft capacity, flight endurance, continuous crew duty duration, and permission to fly over a country's airspace. Given a specific problem, DAKOTA provides a robust interface for the scheduler to employ in interacting with the underlying databases of available airports and other information [6], and the resource allocation tools. This report describes the operational use and design of DAKOTA, and explains the software engineering choices that were made. Details of the extended set partitioning optimization model, the underlying database engine and Structured Query Language (SQL) extensions, and the tutorial and user manual materials are provided in other reports on the DAKOTA system.

1.1 Focus of the Research

A prominent thrust of the research concerns the user-centered design of the software tools.. A truly user-friendly tool is one that requires the user to learn only a few new concepts and easy-to-remember and meaningful keywords in order to get started. The tool should be powerful, allowing the user to solve complex problems by providing assistance in multiple ways, including graphically and statistically, and by providing a way to compare candidate solutions to choose the best one. The greatest promise of a functionality-rich scheduling system is offering new and powerful methods of solving problems. Without adequate system support, the chances of a user finding an optimum solution for a given problem are small. Section 6, discusses the features in the MAP, SCHEDULE and CANDIDATE SOLUTION screens and how the features help in solving scheduling problems. User friendliness also encompasses generating reports in the form of tables, geographic maps, and graphical charts obtained within minutes.

There are two ways to schedule a request; the first is by human schedulers and the second is using the schedule optimizer. The terms scheduler and user in this report refer to human schedulers, unless explicitly described as the automatic scheduler, which refers to the optimization algorithm for scheduling. There are many flexibilities and advantages to providing both manual and automatic scheduling and allowing them work together. For exammple, using automatic scheduler, one could schedule requests and later adjust and tune the schedule using the manual scheduling tools.

Meaningful and helpful error messages contribute to user friendliness, as does a high

quality on-line hypertext help. In some applications, it may be possible to produce computer programs that are so well designed and tested that no user assistance is ever needed. This is seldom achieved in practice because the human ability to understand and predict possible events and outcomes is limited. Help features provide insurance against less-than-perfect design [20]. With a hypertext based help system, the help system author creates the document to fit the information, instead of forcing information into an arbitrary structure. In an off-line document, the document reader must go to a different page or book whenever encountering such things as cross-references, bibliographic citations, glossary terms, and footnotes. In hypertext, by clicking on the hot-spot, the reader can go to a different topic temporarily and return back to the original topic. Hypertext models associative thinking, which closely resembles human idea processing, by creating a network of nodes and links, allowing for three-dimensional navigation through a body of information. Therefore, hypertext tends to easy to learn, understand, and remember.

We also describe the use of higher level software tools in the development of DAKOTA. One such tool is a User Interface Management System (UIMS) called DEC-VUIT. It enables the screen designer to quickly prototype screens. When the screens are to be integrated with other components of the scheduling system, they must be converted to C and embedded within native C source code for the other portions of the system. To accomplish this task, we developed a compiler that parses and generates C source code from the user interface specification language created from DEC-VUIT.

Finally, there was a significant effort to integrate the various sub-systems of the aircraft scheduling software environment. There are many components: the manual scheduling, user interface, database, database interface, help system, UIL2C compiler and optimizer. Each of these components is made up of many sub-components. The approach was to support parallel development of these sub-components, then couple them together. This report describes the manual scheduling, user interface, help system, UIL2C compiler and how other sub-components are invoked in addressing the aircraft scheduling problem.

1.2 Need For the Graphical User Interface (GUI)

There are many computer applications which prototype the work flow of a domain. The success of such systems depends on how accurately and effectively those computer applications represent the domain [1]. Many are not successful in achieving the goal of effectively serving the end-user.

The design of DAKOTA is driven by the end-user. It is built with features that the domain user, in this case human schedulers, need in the scheduling process. For example, the map screen displays a map of the world, with the scheduled mission and unscheduled travel requests explicitly presented. This screen helps in visualizing the problem and increases the productivity of the scheduler. The interactive computer graphics presented are designed to permit an individual with little or no training in scheduling to develop a good schedule in a short period of time. The interactive graphics package provides a bridge between the problem environment and the models and data.

1.3 Development Environment Specifications

This section lists the hardware, software, and functional requirements of the DAKOTA. Hardware specifications, provides the application systems hardware platform requirement.

Software specifications, identifies the programming environment and tools required to develop the application system. The functional requirements are the specifications of the final system, which provide the guidelines during the design and development phases.

1.3.1 Hardware Requirements

The hardware requirements for DAKOTA are as follows:

- Provide the ability to handle a client-server configuration
- Provide the ability for more than one scheduler to work with the application system (i.e., multi-user support)
- Display multiple windows on the display device

1.3.2 Software Requirements

The DAKOTA software requirements are as follows:

- Support multiple picture formats, including X PixMap, .and gif, and provide tools to create and edit pictures
- Provide libraries for handling multiple types of display objects, including menus, pushbuttons, drawing areas, text fields, and scroll bars.
- Provide facilities for dynamically changing the display attribute of the display objects.

1.3.3 Functional Requirements

There are various functional requirements of the target system, delineated in the following paragraphs.

Graphical User Interface:

- Provide efficient data entry and retrieval screens which involve the use of graphical user interfaces
- Provide the ability to run multiple computer sessions simultaneously (e.g., a scheduler being able to view request details when scheduling)
- Provide efficient and intuitive navigation between screens for both sophisticated and unsophisticated users
- Provide the flexibility to incorporate a variety of data input methods (i.e., keyboard, mouse, etc.)

Map Display: The system should display a map, enabling the visualization of scheduled mission paths. The user should be able to perform zooming and scrolling operations on this window and should be able to selectively hide and view information in the map.

Saving and Loading of Schedules: Scheduling is a complex paradigm, and complexity increases with the number of requests, aircrafts and constraints imposed by the requests from clients. A typical mission scheduling could be a time-consuming job, taking a major part of a human scheduler's time and effort. The system should provide schedulers a way to save their work to a permanent storage and continue later by retrieving from the repository.

Report Generation: The user should be able to take a hard copy of the generated schedules in the form of a professional report. As postscript printing is becoming an industry standard, DAKOTA should be able to generate postscript output.

Customization: The user should be able to choose the shape and color of various objects, depending on likes and needs. The user should be able to give default values for objects and be able to store them in the repository. Once the screen comes up later, the system should be able to remember and use the setting given by the user in the past.

On-line Help: On-line help should be designed with a novice user in mind. On-line help should be consistent with the hard copy user manual. Many different types of navigation facilities should be available to the user to facilitate finding a topic of interest with ease and in the shortest time.

Extendibility: The system should be extendible so that other automated scheduling systems could be replaced or made to work in a cooperative manner within DAKOTA. Sufficient modularity should be designed into the system so that later it could be made to work with other Relational Database Management Systems(RDBMS).

Considering the system requirements we have decided on developing DAKOTA in a DEC hardware, running a variant of the UNIX operating system called Ultrix V4.3. X Window system is the development of a GUI environment for the whole system. This system satisfies all the software and functional requirements mentioned above.

1.4 Organization

This report presents the DAKOTA system from the user interface and software design perspectives. The organization and content of subsequent sections are described in this section. This report contains seven sections and four appendices.

- Section 1 describes the purpose of the DAKOTA system, the associated design considerations, and the graphical user interface for scheduling. This section also presents the focal point of this work and system requirements in terms of hardware and software.

- Section 2 introduces the nature and definition of the problem, architecture of the application system, and context related to the literature. Since the main thrust is given to the user interface design, this section builds a foundation for the next section which provides analyses of the user interface issues in detail.

- Section 3 explains the user interface issues which influenced the design of DAKOTA. This section lists the characteristics of importance in designing the human computer interface. Subsections include a detailed description of each of those characteristics and present example screens showing how they are adapted in DAKOTA.

- Section 4 describes manual scheduling with real-world examples and screens to provide an intuitive method of scheduling in the manual mode. This section also discusses how automated and manual scheduling work cooperatively, thereby taking advantage of the best of both worlds.

- Section 5 presents the overall design of the system. The design process uses software engineering concepts in the design and development of DAKOTA and its sub-systems. This section contains entity-relationship diagrams (ERD) and data-flow diagrams (DFD) which describe the data modeling and process modeling. This section also covers validation and data integrity checks for on the user data before populating the database.

- Section 6 explains some of the algorithms used in manipulating graphical items such as zooming of a world map, viewing and comparing solutions generated either manually or automatically, and helping choose the best solution possible.

- Section 7 concludes the report by summarizing user interface issues and design recommendations and how they are adapted in DAKOTA. It also lists the enhancements in terms of features and functionalities that could be added to DAKOTA.

- Appendix A discusses master data management in detail. It also shows data entry screens used for querying and manipulating airport atlas records, passenger details, and aircraft records.

- Appendix B contains a data file format, sample data, and a screen interface for saving and loading schedules in permanent storage.

- Appendix C presents a user manual for the hypertext help system. It also includes sample hyper-help screens, an input data file format, and the data model schema.

- Appendix D presents a user manual for the UIL to C compiler. It also includes a sample input data file format and sample output file.

2. Background and Nature of the Problem

Studying actual work activity leads to an understanding of how the participants accomplish their tasks. Based on that understanding, design implications for tools to support domain user activity are identified and embodied into prototype tools [3]. In Section 2.1 we describe the activities that influence the scheduling problem at USAFE. An overview of the system, developed to solve user needs, is given in Section 2.2. Section 2.3 consists of descriptions of tools that directly and indirectly assisted in the development of DAKOTA.

2.1 Definition of the Problem

The definition of the executive scheduling process is driven by five main entities. They are the human schedulers, clients, flying requests, resources, and crews. These entity characteristics and constraints define the scheduling paradigm.

2.1.1 Clients

Clients are both the originators of the data flow and the beneficiaries of the output from the system. They originate the data by filing a request to fly from one place to another. The system analyzes the data and schedules a request for a flight. The flight squadron executes the schedule produced from the software system.

2.1.2 Nature of the Requests

Requests originate from clients, and each of those requests has different characteristics and constraints. A typical request consists of a sequence of origin and destination locations with start and end times of the trip. Each of these request origin/destination pairs is called a request leg. Every request leg has associated time window information, called the departure time and arrival time window. This time window provides room for human schedulers and automated scheduling algorithms to execute and produce an optimized schedule. A time window can be designated as soft or hard, depending on whether it can or cannot be violated.

A request is originated by a member in a group or a single person who wishes to fly from one place to another. The contingent size may vary, and an individual may join or leave the group between two request legs. In addition, a request can have additional constraints concerning such things as the need for a specific aircraft, allowance of combining of requests. etc. Cancellation of the request can also occur; therefore, the system provides ways for unscheduling request(s) that have been previously scheduled. There could also be unforeseen request constraints; thus, a means for recording the constraints must be provided. Schedulers refer to these comments or remarks while scheduling that request.

2.1.3 Resources

The resource in this scheduling environment refers to aircraft. There are 20 aircraft in the fleet falling into seven different aircraft types. Each type has its own specifications; they are the air speed, capacity, endurance (continuous time in flight without refueling), maintenance schedule, and the budgeted flight time per year. Typical air speed of the flight in the fleet is between 100 to 500 knots, capacity is from 5 to 20 seats with few of upto 50, and

endurance of the flight is between 2.5 to 10 hours.

The system provides ways to add new aircraft types and modify specifications of existing aircraft types. It also provides ways to add and remove individual aircraft to an aircraft type. A 5-digit number, called tail number, uniquely identifies an aircraft. The tail number is not sharable, once assigned to a mission. The system should not let a tail number be scheduled for the time window it is already scheduled to a mission.

2.1.4 Crew Management

The crew or pilot executes a schedule. Actual assigning of crews to a mission is handled by the flight squadron; it is not presently supported by the DAKOTA system. De-coupling crew assignments and mission scheduling provides maximum flexibility for assigning any available crew eligible to fly the aircraft at the time of the mission. Also while scheduling, there is no need to be constrained by the availability of any particular crew. However, crew duty day, a crew-related constraint, affects the scheduling process. A mission should not exceed a crew duty day, which is 16 hours including 2 hours pre-flight. A crew duty day is the time when a pilot reports for duty until he or she completes the mission. Hence, flights over 14 hours should include a crew rest period of 15 hours. This procedure applies to most of the aircraft types in the fleet. The system captures these violations during scheduling and reports them to schedulers.

2.1.5 Schedulers

Scheduling involves assigning an aircraft or resource to a request. This process has to consider all the constraints imposed by the clients, requests, resources, and crews. The output from this process is a set of mission(s) with mission legs (origin/destination pair). It can be done either manually by human schedulers, automatically by the automated optimizer, or with a combination of both human schedulers and the optimizer.

Mission scheduling can be a prolonged process. Therefore, mechanisms need to be built into the system to save the current work and load it later to continue from where it was left. Also, changes to a scheduled mission can happen up to the last minute, just before it is executed by the flight squadron. The system provides ways to change any mission-related information. DAKOTA provides mechanisms to generate more than one solution for a given problem and to compare and contrast solutions statistically and graphically.

2.1.6 Objective

The objective of the operation is to service clients to their satisfaction with optimal usage of resources. This goal implies that the routes of the aircraft missions must be efficient, minimize aircraft travel time, and that the selection and sequencing of requests supported by a mission facilitate high aircraft utilization. Given this objective. the next section provides an outline of DAKOTA as it was developed to meet system requirements.

2.2 System Overview

The scheduling software system has four main components; the presentation layer, the database layer, the help sub-system and the algorithmic layer for automatic scheduling. Figure 2.1 depicts the architectural overview of the software system. There are several screens to manipulate data in the scheduling paradigm. A database [6] is used to maintain data that are referred in the scheduling process. Interaction with the database takes place using an API (Application Program Interface) provided by the database. To perform manual scheduling the presentation and database layer are employed. The automatic scheduler is a high-level layer above the presentation layer, and is a loosely coupled module. The presentation layer invokes the automatic scheduler at the request of the user. All required data by the automatic scheduler goes from the database through the presentation layer. The on-line hyper text help system is a support module for this scheduling software system. It provides the end-user of the application with on-line information of how to use the software system. With this background on the scheduling software system, the subsequent paragraphs give an overview of the features and function of the DAKOTA from the end-user's perspective.

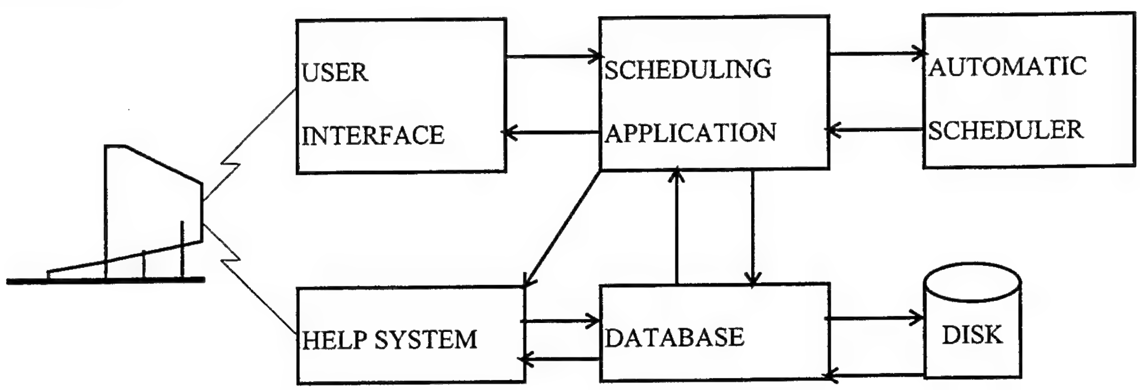

Figure 2.1 Architectural overview of DAKOTA.

The software has a screen interface to support all of the steps involved in mission scheduling. It has screens to manage master data, which includes an airport atlas and passenger and resource/aircraft information. The actual scheduling process starts when entering client request information, using the request screen interface. This request is scheduled into a mission in the scheduling screen. A map screen (see Figure 6.1) is used to view scheduled routes and unscheduled requests. In addition, there are several viewing mechanisms built into the system to enable visualization of the problem and produce efficient schedules. This section will briefly describe the screen interfaces in DAKOTA. It also describes how the application is layered and integrates components such as the automated scheduler, Hyper-Help context sensitive on-line help. It also describes how the UIL2C - UIL to C

converter is used with DEC-VUIT for rapid screen interface prototyping.

2.2.1 Screen Interface

There are eight screens in the DAKOTA system:

- Main Window
- Airport Atlas Information
- Passenger Information
- Aircraft or Resource Information
- Request Data Entry
- Scheduling Screen
- Map Screen
- Candidate Solution Viewing Screen

Main Window: This is the first screen that appears to the user. Other screens in DAKOTA may be accessed from this screen by pressing the appropriate button from the menu bar. The main window also displays the logo of the end-user of the system. There are three master data entry screens: airport atlas, passenger, and aircraft. These screens provide the facility to add a new record, modify or remove an existing record from the database. These are lookup data, referenced during mission scheduling and request data entry.

Airport Atlas: This data entry screen is used to capture all details pertaining to an airport. It includes name of the airport, city where it is located, latitude, longitude and, a four-letter identification code called ICAO. This code is used universally to refer to an airport. DAKOTA uses this code in the scheduling screen; request data entry, and in map screens whenever there is a reference to an airport. This screen has facilities to add a new airport to the database, modify already existing airport information, and delete an existing airport record from the database.

Passenger Information: This data entry screen is used to manage information about persons who have already flown a mission or who are likely to fly a mission. Records in this database keep growing. At some point passengers who are not likely to fly any more missions in the future need to be identified and removed.

Aircraft or Resource Information: This data entry screen is used to manage a heterogeneous fleet of aircraft information. There are five total different kinds of information recorded: type of aircraft, air speed, capacity, endurance, and flying hours. There also could be more than one aircraft of a specific type, identified by a tail number. A separate screen is used to capture all the different aircraft and tail numbers within a aircraft type.

Request Data Entry: This screen is used to input requests for travel and screen is arranged into four sections. The first section is used to enter general information about the

request. Passenger information is entered in the second section, request legs, or a departure/arrival pair, are entered in the third section. Finally, the fourth section is used to enter general remarks and notes about the airlift request that can be used by the schedulers.

Scheduling Screen: Schedulers use this screen to schedule missions. In a scheduling session one or more mission(s) from one or more request(s) could be scheduled. Basically, there are two parts to a scheduling window. The upper half of the screen displays a mission(s) that participates in a scheduling session, while the lower half of the screen displays all the requests that take part in that scheduling session. This screen is mainly used for manual scheduling, but it also has a provision to invoke the automatic scheduler. Both of these methods can be mixed and matched to reach an optimized schedule. There are facilities to save and load a scheduling session to and from permanent storage. A detailed description of the scheduling process is provided in Section 4.

Map screen: This screen enables schedulers to graphically visualize missions and requests on a particular date. The map screen is divided into four separate panes. The upper left pane contains a map of the world with lines corresponding to mission legs and request legs. Missions and requests are color coded to differentiate one from the other. Zoom facilities are available to the user, which enables the scheduler to zoom on a particular area in a map to get a clearly focused view of the area of interest. The upper right pane displays all the unscheduled requests, while the lower right pane displays scheduled missions on a selected date. The lower left pane displays aircraft type flying hour data. This gives an idea of how much of an aircraft type's flying hours are used, as opposed to the budgeted allocation.

Candidate Solution Viewing Screen: This screen looks similar to the map screen, but it displays missions and requests of a scheduling session, as opposed to displaying a day's fleet activity. It also displays activity of aircraft in a scheduling session in the form of a cellular graph. It helps compare two or more schedule sessions and choose the best available solution for a given problem.

There are two main reports generated from this software. First, the mission report, contains the itinerary, passenger list, and revision history for a mission. Second, the daily reports, contain a day's schedule. This includes name of the leading passenger in every contingent, itinerary of all missions on a given day, and the aircraft type and tail number.

2.2.2 Sample Manual Scheduling

In this section we explain the scheduling process with a sample problem. The example demonstrates how the user can schedule these requests into missions with efficient use of aircraft and comply with the clients request constraints. Following is a list of requests populated into the database using the request data entry screen:

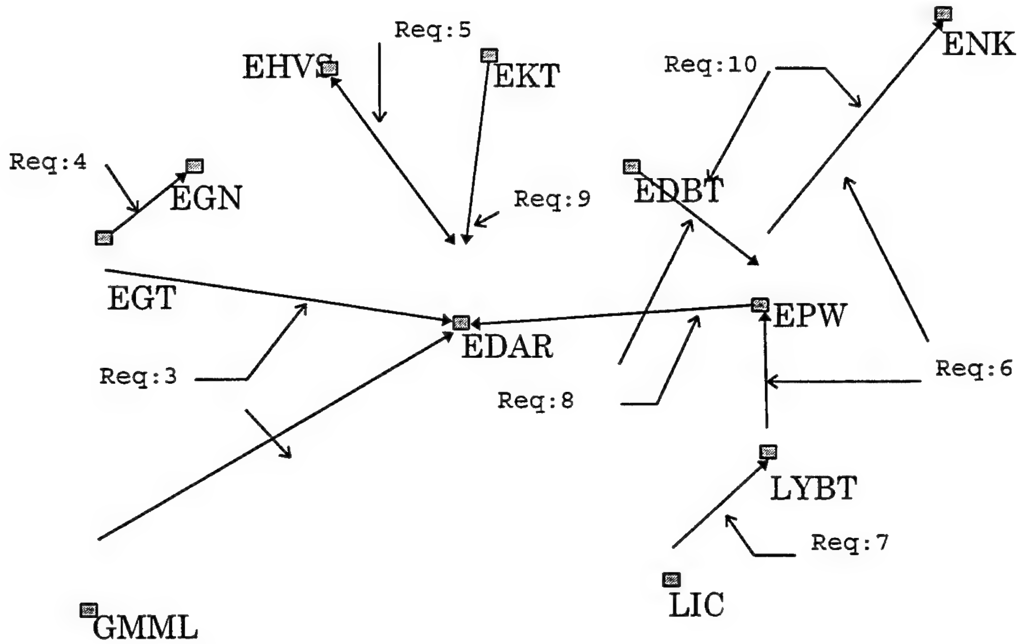

Figure 2.2 Requests as seen in map screen.

Req	Origin		Pickup Time		Destination		Pickup Time		PAX
Id	ICAO	Date	Early	Late	ICAO	Date	Early	Late	#
3	GMML	5/Apr/95	0400	0600	EDAR	5/Apr/95	1100	[1200]	2
	EGTE	5/Apr/95	0900	1030	EDAR	5/Apr/95	1100	[1200]	2
4	EGTE	5/Apr/95	0250	0500	EGNB	5/Apr/95	0700	[1000]	2
5	EDAR	5/Apr/95	[0650]	1230	EHVB	5/Apr/95	1200	[1300]	3
	EHVB	5/Apr/95	[1150]	1730	EHVB	5/Apr/95	1700	[1800]	3

6	LYBT	5/Apr/95	0835	1050	EPWA	5/Apr/95	1030	1230	2
	EPWA	5/Apr/95	1130	1330	ENKR	5/Apr/95	1400	[1700]	2
7	LICD	5/Apr/95	0630	0820	LYBT	5/Apr/95	0800	0930	2
8	EDBT	5/Apr/95	0920	1040	EPWA	5/Apr/95	1005	1230	2
	EPWA	5/Apr/95	1200	1330	EDAR	5/Apr/95	1330	[1430]	2
9	EKTS	5/Apr/95	1850	2150	EDAR	5/Apr/95	2200	[2300]	3
10	EDBT	5/Apr/95	0450	1145	EPWA	5/Apr/95	1030	1230	3
	EPWA	5/Apr/95	1200	1415	EPWA	5/Apr/95	1400	1700	3

Table 2.1 Sample requests used to demonstrate manual scheduling.

Dates are given in day/month/year format; times are railway time; '[' and ']' surrounding a time indicate hard time bound.

Even though we have eight different requests, some legs overlap. If it happens that the time windows are flexible enough (i.e soft time), we can schedule those eight requests in missions fewer than eight. Following are the steps in solving a scheduling problem:

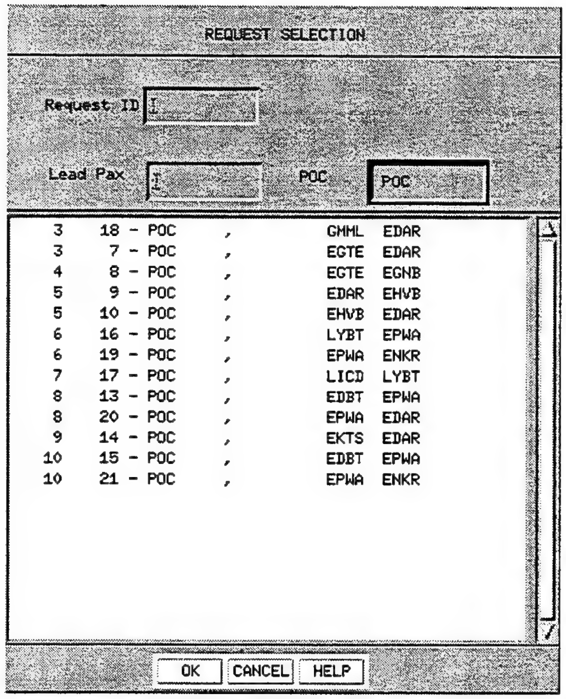

Figure 2.3 Request selection dialog box.

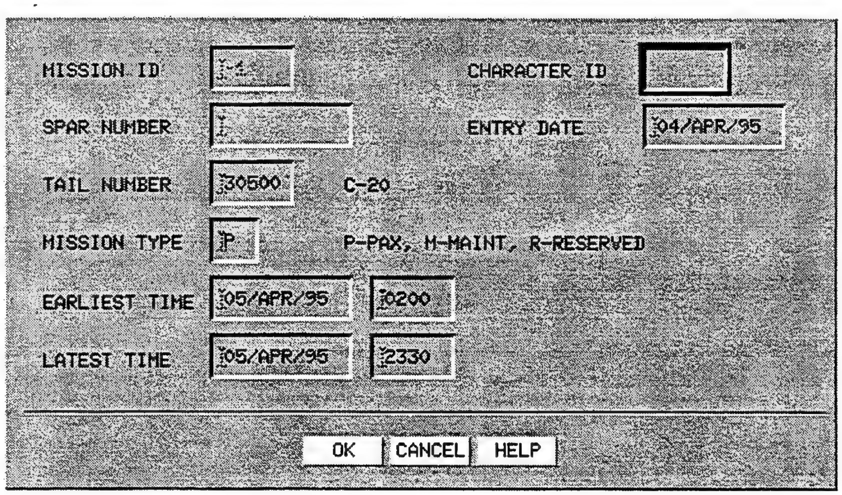

Figure 2.4 Mission Creation dialog box.

The first step in problem solving is to define the problem. This is done by selecting unscheduled request(s) in the request selection dialog box. Clicking DEFINITION->SELECT_REQUEST from the pulldown menu in the scheduling screen will pop up the request selection dialog box. A sample request selection dialog box is in Figure 2.3, showing selected unscheduled requests legs which are on the list from Table 2.1.

The second step in scheduling is to create missions or, in other words, block aircraft/resource time for scheduling. Selecting DEFINITION->SELECT_MISSION from the pulldown menu in the scheduling screen will pop up the mission creation dialog box. Figure 2.4 shows a mission that is created. After entering all text fields, pressing OK will create the mission with a negative identifier. As more missions the system will automatically assign an ID below 0 in descending order. A positive mission identifier is assigned to a mission when the user chooses to save to database. This type of mission number assignment is done to make sure there is no gap in the mission sequence number.

The third step in the scheduling process is to create mission legs such that more requests can be accommodated in fewer missions possible without violating request, aircraft and crew constraints (as mentioned in Section 2.1). Mission legs can be created by following steps given in Section 4. Looking at the unscheduled request legs, the user can accommodate in four missions. A list of mission legs created is shown on Table 2.2.

The fourth step in the scheduling process is to associate unscheduled request legs to missions. First, click on the SCHEDULE button in the request work area, and then select an unscheduled request leg to be scheduled. The system will highlight mission leg(s) that can accommodate selected unscheduled request legs. There could be more than one combination of mission leg(s) that could fly a request leg. The NEXT and PREV button will show successive combinations of mission leg(s) that could fly a request leg. Press OK when a particular mission leg(s) combination is acceptable. Continue this step until all unscheduled request legs are scheduled. A sample scheduling with example scheduling data is shown in Figure 2.5.

Finally, save the work to permanent storage by selecting FILE->SAVE_TO_DB in the pulldown menu.

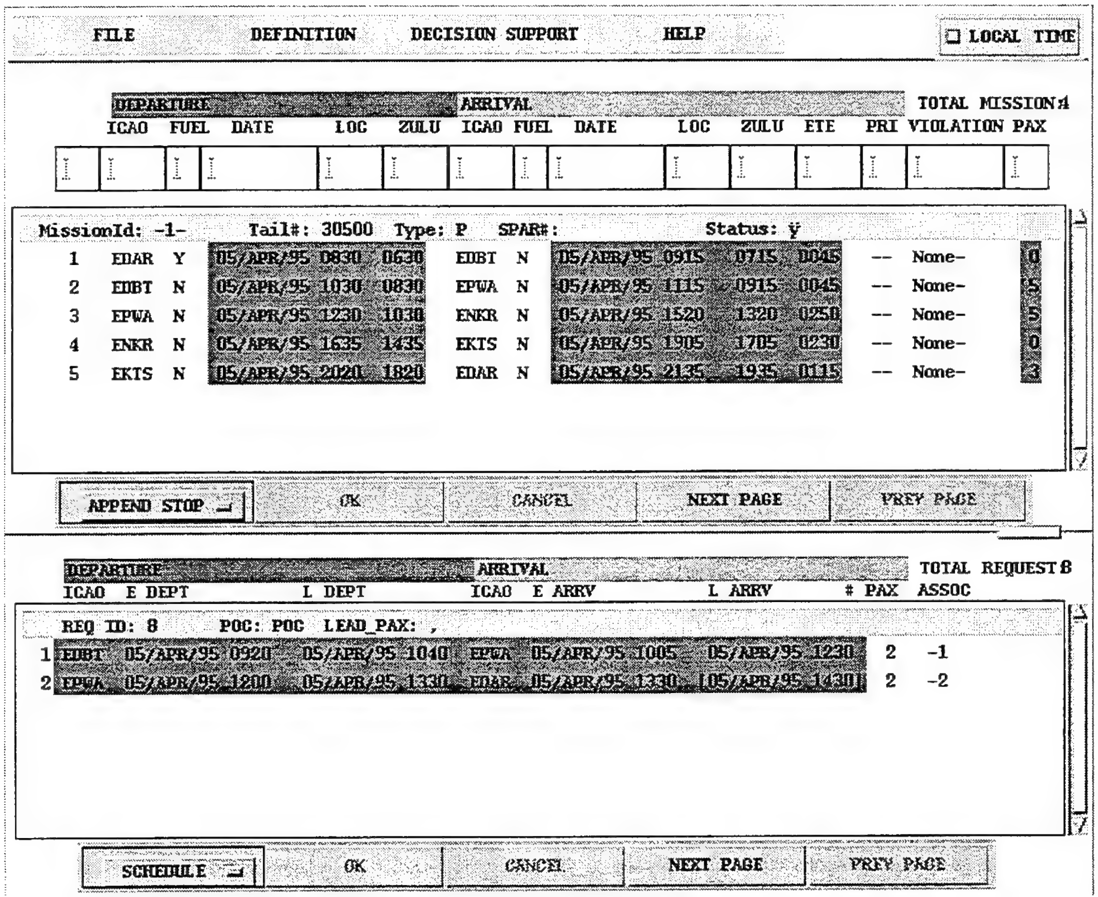

Figure 2.5 Scheduling screen with sample data.

Miss	Origin				Destination				PAX	Violation
Id	ICAO		Date	Time	ICAO		Date	Time	#	
30500	EDAR	Y	5/Apr/95	0830	EDBT	N	5/Apr/95	0915	0	-none-
	EDBT	N	5/Apr/95	1030	EPWA	N	5/Apr/95	1115	5	-none-
	EPWA	N	5/Apr/95	1230	ENKR	N	5/Apr/95	1520	5	-none-
	ENKR	N	5/Apr/95	1635	EKTS	N	5/Apr/95	1905	0	-none-
	EKTS	N	5/Apr/95	2020	EDAR	N	5/Apr/95	1935	3	-none-
40084	EDAR	Y	5/Apr/95	0400	LICD	N	5/Apr/95	0615	0	-none-
	LICD	N	5/Apr/95	0730	LYBT	N	5/Apr/95	0915	2	-none-
	LYBT	N	5/Apr/95	1030	EPWA	N	5/Apr/95	1140	2	-none-
	EPWA	N	5/Apr/95	1255	EDAR	N	5/Apr/95	1420	2	-none-
30502	EDAR	Y	4/Apr/95	2300	GMML	N	5/Apr/95	0320	0	-none-
	GMML	N	5/Apr/95	0450	EGTE	N	5/Apr/95	0945	2	-none-
	EGTE	N	5/Apr/95	1100	EGNB	N	5/Apr/95	1135	7	-none-
	EGNB	N	5/Apr/95	1250	EDAR	N	5/Apr/95	1455	4	-none-
40085	EDAR	Y	5/Apr/95	0730	EHVB	N	5/Apr/95	0805	3	-none-
	EHVB	N	5/Apr/95	1230	EDAR	N	5/Apr/95	1305	3	-none-

Table 2.2 Missions for requests in Table 2.1

2.3 Justifying Need for a GUI Environment

GUI applications acts as a bridge for the user to communicate with the back-end processing module, hide complexity of the processing, and provide an easy method to present and solve problems. Without a front-end GUI, the scheduler has to have sufficient knowledge about a problem-solving domain, processing complexity, data storage, and manipulation [15]. In other words, the GUI permits the user to abstract from task details and concentrate on higher level issues. The GUI also helps the user to better visualize the problem, suggest options, compare solutions, and edit problem definitions with ease. The interface checks a user's authority to perform certain tasks, or it may require confirmation before performing potentially costly or irrevocable actions [16]. It integrates various software technologies like client-server computing, complex data storage and retrievals, heuristic algorithms, etc. to provide solutions for end-user problem.

The scheduling environment handles a huge amount of data. It has approximately 10,000 different airport information, hundreds of passenger details, around 20 aircraft in the fleet, and up to 20 requests for schedules coming in every day. Managing this huge data without a user interface is tedious. An interface allows a scheduler to perform tasks faster and keeps the user's interest over longer periods without rest, by using colorful displays and intuitive screens, as scheduling is a routine task. Considering the above advantages and the volume of data handled in the system, there is strong preference for using a graphical user interface in aircraft scheduling software system.

2.4 Context Relative to Literature

As this report concentrates on the user interface all of the user interface codes are developed in X window systems. It becomes relevant to briefly talk about X concepts and architecture in the next paragraph. DEC-VUIT, a User Interface Management System (UIMS) from Digital Equipment Corporation, is used to quickly prototype screens in DAKOTA. Its features and functions are briefly described in Section 2.4.2. All pictures in the on-line help sub-system are displayed using XPM library calls. XPM picture formats and library calls are briefly enumerated in Section 2.4.3.

2.4.1 X Window Systems

The X window system is an industry standard software system that allows programmers to develop portable graphical user interfaces. One of the most important features of X is its device-independent architecture. The Massachusetts Institutes of Technology (MIT) initially proposed and developed X, and it later became the responsibility of the X Consortium. The X window system is complex, but it is based on a few premises that can be quickly understood. The first and most obvious thing to note about X is that it is a windowing system for bitmapped graphics displays. In bitmapped graphics, each dot on the screen (called a pixel or picture element) corresponds to one or more bits in memory. In X, a display is defined as a workstation consisting of a keyboard, a pointing device such as mouse, and a screen. The second thing to note is that X is a network-oriented windowing system. The program that controls and manages a display is known as the server. The server acts as an

intermediary between user programs called clients or applications, running on either the local or remote systems [9]. The server performs the following tasks:

- Allows access to the display by multiple clients
- Interprets network messages from clients
- Passes user input to the clients by sending network messages
- Performs drawing-graphics on behalf of the clients
- Maintains complex data structures, including windows, cursors, and fonts, and shares those among clients

The X window system provides an event-driven programming paradigm. An event can be initiated by the user or the system. Events include user input (key-press, mouse click, or mouse movement etc.) as well as interaction with other programs. For example, if an obscured portion of a window is exposed when another overlapping window is moved, closed, or resized, the client must redraw it. All the events are first trapped by an X server and passed on to an X client through a queue. Events are placed on a queue in the order they occur and usually are processed by clients in that order. Figure 2.2 depicts the client-server model of the X window system.

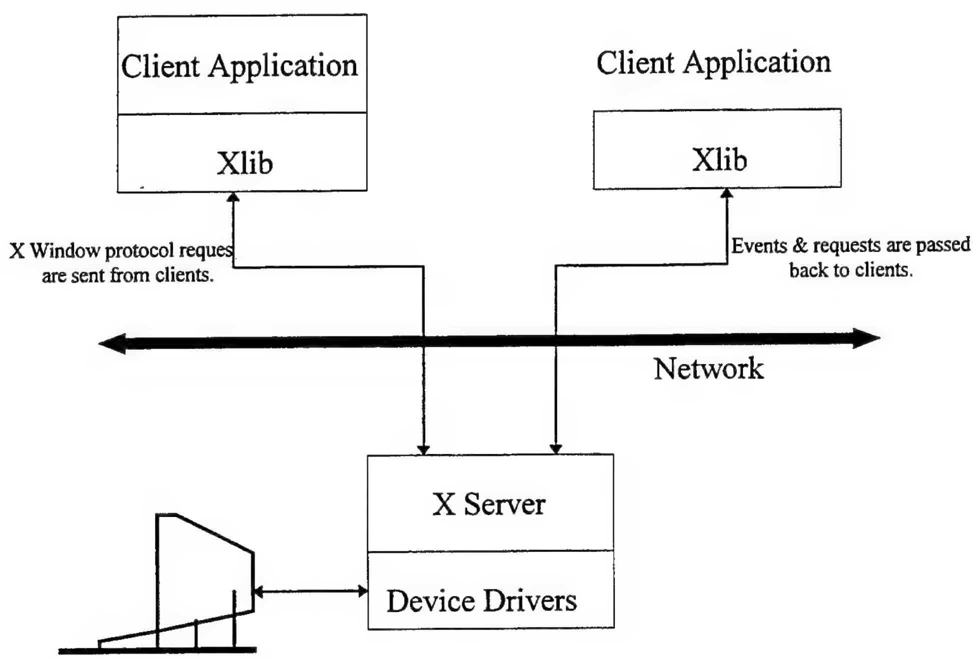

Figure 2.6 X client-server model.

The client uses the Xlib layer to communicate with the server using a protocol. The protocol used in such communication is called X protocol. The X protocol specifies what makes up each packet of information transferred between the server and client. There are four types of packets transferred via the protocol: requests, replies, events, and errors. A

request protocol is generated by Xlib and sent to the server. A protocol request can carry a wide variety of information, such as a specification for drawing a line or an inquiry about the current size of a window. A protocol reply is sent from the server to the client in response to certain requests. Not all requests are answered by replies, only the ones that request information. An event is sent from the server to the client and contains information about a device action or about a side effect of a previous request. A protocol error tells the client that a previous request was invalid. An error is like an event, but it is handled slightly differently within Xlib. So far X concepts and architecture were discussed. The next paragraph discusses X Toolkits (Xt) and the Open Software Foundation Motif (OSF/Motif), which are layers over Xlib developed for the programmers' convenience.

The Motif is a layer over X Toolkit (Xt), which in turn is layered on top of Xlib, thus extending the basic abstractions provided by Xlib. Although Xlib provides the fundamental means of interacting with the X server, developing a complex application using Xlib is a formidable task. X Toolkit Intrinsics (Xt) provides a higher level programming interface with objects known as widgets, mechanism for dispatching and handling events, and an easy way for handling widgets geometry. Xt supplies the substrate for creating a set of widgets responsible for specific aspects of a user interface. Motif uses the Xt substrate to build base classes and specialized subclasses of widgets for variety of purposes [9]. Apart from this, Motif adds a number of features that are of convenience to the application programmers in building complex applications. DAKOTA uses the facilities provided by Motif, Xt, and Xlib layers. Figure 2.3 depicts the layered architecture of the X window system.

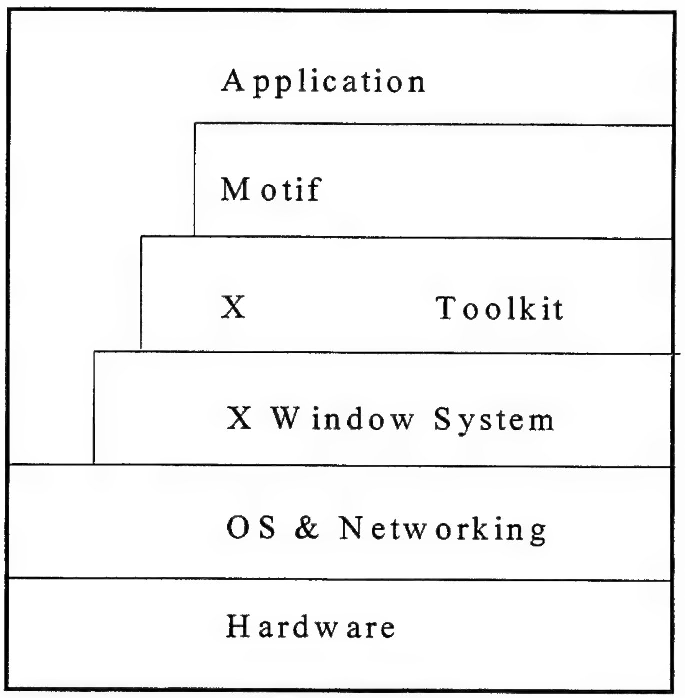

Figure 2.7 X window systems layers.

2.4.2 DEC-VUIT

DEC-VUIT is a User Interface Management System (UIMS) from Digital Equipment Corporation (DEC). DEC-VUIT was used in developing all screens in DAKOTA. It is a graphical tool to quickly paint screens by point and click. Once a screen is painted, an ASCII file can be generated containing User Interface Language (UIL) scripts from the DEC-VUIT tool. The UIL is a specification language for describing the screen attributes such as width, height, color, font, etc. for display objects such as menus, dialog boxes, labels, push buttons, etc. The UIL also specifies the routines to be called when the interface changes state as a result of user interaction. A UIL containing the list of objects and object resources is compiled into a UID file, using the UIL compiler. The contents of the compiled UID file can then be accessed by various Motif Resource Management (MRM) functions from within an application program to query or modify the characteristics of an object.

Although there are many advantages to using DEC-VUIT and its UIL file, there are several disadvantages, such as no support for dynamically modifying callbacks for widgets. Since DAKOTA is required to dynamically modify widget callbacks, a solution for this problem was needed. Therefore, a UIL2C compiler was developed, for compiling a UIL script into equivalent C code. This C code helps improve the display performance of the application and also allows the user to dynamically modify widget callbacks.

2.4.3 XPM - X PixMap

The DAKOTA displays pictures in the on-line hypertext help sub-system. The pictures are stored in a separate file in XPM format [10]. The XPM library is a set of calls enabling picture data to be read from an ASCII file and displayed on the screen. There are quite a few picture formats available in practicality, but XPM was chosen because for the following reasons:

- XPM picture data files are machine independent. These files have a sequence of ASCII characters which can be easily transported either through a magnetic medium or through e-mail.

- Picture format presents a C syntax, in order to provide the ability to include XPM files in C and C++ programs. As DAKOTA was developed in C language it became natural to use XPM.

- There are a rich set of function calls available to manipulate XPM pictures.

- Editing a picture can be done using any text file editor, as the XPM picture file is ASCII.

The following is a template for an XPM picture file. There are basically seven sections to an XPM picture file:

- Header line

- Declaration and beginning of assignment line

- Values

- Colors

- Pixels
- Extensions
- End of assignment

Header Line: This line contains the comment keyword /* XPM */.

Declaration and Beginning of Assignment Line: This line is composed of "static char * <variable_name>[] = {". The values section is a string containing four or six integers in base 10 that correspond to: the pixmap, width, height, number of colors, and number of characters per pixel.

Colors: This section contains as many strings as there are colors, and each string is as follows:

chars - {<key> <color>}+

> where chars is the <chars_per_pixel> length string representing the pixel, <color> is the specified color, and <key> is a keyword describing in which context this color should be used.

Pixels: This section is composed of <height> string of <width>*<chars_per_pixel> characters, where every <chars_per_pixel> length string must be one of the previously defined groups in the <colors> section.

End of Assignment: This section ends the formats with a closing brace "}".

The XPM library provides a set of Xlib-level functions which allow the programmer to deal with images, pixmaps, and the XPM file. In DAKOTA the XpmReadFileToPixmap() function is used, which creates a pixmap from XPM data file.

3. USER INTERFACE DESIGN ISSUES

DAKOTA incorporates several Graphical User Interface (GUI) design principles. This section summarizes GUI design principles, illustrating how GUI is employed in this package. Many of the user interface design concepts are obtained from style guides [7] [8].

3.1 User Orientation

The DAKOTA system carefully designed to solve the needs of the end-user. Users should feel they are in total control of the application, not just following a sequence of steps provided by the system [11]. Changes can be made, and the capability of undoing the work later is provided. The user interface avoids modes that severely restrict the interactions available to the user at any given time. Apart from modes there are other characteristics that influence the user interface design, which are discussed in this section.

3.1.1 Direct Manipulation (DM)

The interface gives users direct and intuitive ways to accomplish their tasks. The object-action paradigm supports this principle. The user performs tasks by selecting an object (such as an icon, a window, or some text) and then selecting an action (such as move, update, or delete) for that object. Manipulating objects directly, although not appropriate in all situations, is often easier than typing complex commands [16]. For example, it is much easier to move a window by dragging it with the mouse than by visually estimating new coordinates and then typing them into a dialog box. In DAKOTA, users can get more information about a mission by directly clicking on one of the legs in the map screen, instead of typing the mission identifier in a dialog box.

3.1.2 Consistency

In DAKOTA objects and elements are consistent within and among applications [8]. There are two points in building a consistent user interface. First, the user interface should be consistent with its actual real-world process. This helps the user of the application to easily adapt and learn the computer-based application. Second, each user interface screen should be visually and functionally consistent. This helps both the designer and the developer produce a well-designed, reusable, standard interface object. It also helps the user to become acquainted with all parts of the application with ease as every segment of the application has a similar look and feel interface element. In the DAKOTA schedule screen, scheduling is completed by first clicking on a request leg, followed by selection of mission leg(s). In the real world while scheduling manually, each unscheduled request leg in a request is associated with mission legs, one by one on paper, to achieve the final goal of solving the problem. Here, how one would think and act is simulated in solving a problem manually, by providing an efficient user interface and suggesting all possible mission legs where a request leg would fit. Another example is that the map screen enables visualizing fleet activity by day. In the previous systems, schedulers had to mentally visualize a day fleet activity. In DAKOTA, the system displays all the missions, unscheduled requests, and unused resources in the screen. Therefore, the users can concentrate on actually generating an efficient schedule.

3.1.3 Clarity

The application interface should be visually and conceptually clear. Visual elements should be immediately comprehensible, ideally because they relate to real-world analogs, and should be arranged so that their functions are comprehensible. Interface text should be clear, unambiguous, free of jargon, and consistently used to refer to the same meaning. DAKOTA uses the scrollbar in all screens wherever we need to show more information than could fit on the screen. A scrollbar has two arrows, one at each end, and a slider in the middle portion of the scrollbar. The position and size of the slider indicate which portion of data in the screen is currently visible. Space on either side of the scrollbar indicates amount of data before and after the data that are currently visible on the screen. This type of user interface object is immediately comprehensible to the user without much explanation. It is also conveys the idea of the developer, that is, how much data the end-user could see on the screen.

3.1.4 Groupings

Grouping of functionally related interface objects greatly enhances the clarity of the user interface. Grouping reduces search time for required information in a complex user interface environment. In DAKOTA, all mission-related functionalities are grouped under the MISSION pulldown menu on the scheduling screen.

3.1.5 Feedback

The DAKOTA user interface is highly interactive. The user receives immediate feedback for actions within the application. If the user of the application selects an interface or data object with the mouse, the application provides visual feedback that the object is selected. The response is could be graphical, textual, auditory or some combination. Clicking on a row using the mouse in the text editing screen changes the color of the clicked row. This visually assists the user in knowing that the clicked row is the row that the user intended to select for editing. DAKOTA changes the background color of the selected row from white to cyan.

3.1.6 Compatibility

It is important that input and output be compatible. For instance, in the map screen a text field accepts values in date format. Changing the date in the text field is possible either by typing a new date or by clicking the up arrow or down arrow button to increment or decrement dates using the mouse. Compatibility also refers to the presentation of data being compatible with the environment where the application is used. For instance, different countries follow different date formats. The United States follows mm/dd/yy, where mm, dd, and yy refer to month, date, and year; but European countries follow dd/mm/yy. In DAKOTA, these types of situations are handled by letting users specify the date format in the startup configuration file.

3.1.7 Flexibility

Flexibility refers to the providing alternative ways for the user to achieve a certain goal. For example, the aircraft data entry screen is used to both view and maintain aircraft information. The aircraft screen can be invoked in many ways. The screen can be selected from the pulldown menu in the Main Control Window(MCW), and it can also be selected from the map screen. It is logical to provide a way to get to all data entry screens from the MCW, and it can be useful for the schedulers to know airport details when they are viewing a list of missions and requests in the map screen.

A flexible user interface also refers to capability of the user changing the program to better suit individual needs and preferences. It is difficult for a user interface designer to foresee all the user preferences while designing a user interface. In DAKOTA, this is partially addressed by providing a programmable user interface, also called a customizable user interface. This interface provides a means by which some of the final design choices can be delayed until the final commitments to the user are established[16]. However, too many choices creates the possibility of spoiling the aesthetics of the user interface. DAKOTA provides a customize button to the user to select options for changing the map display or the mission display. The map display option enables the user to customize the display of the data

on the map. Customizing is done by clicking on the check box shown in Figure 3.1.

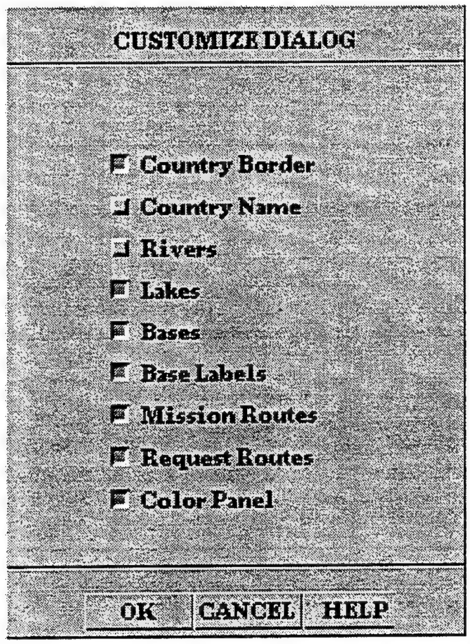

Figure 3.1 Color selection dialog box.

The Map can be customized to display the following information on the screen in any combination:

- Country Border
- Country Name
- Rivers
- Lakes
- Bases (shown as small circles; large circle corresponds with main base)
- Base Labels (4-character length code, i.e., ICAO)
- Mission Routes (each mission in shown in different color)
- Request Routes (differentiated from missions by dotted lines)
- Color Panel (shows color associated with a mission)

3.1.8 Usability

Usability of the application can be improved by contextual field research, a growing area in Human-Computer Interaction (HCI). Usability emphasizes the importance of context in

understanding usability issues. A more focused investigation of the existing systems to uncover the winners and losers in functionality and interface designs will help design the new system [12]. This investigation addresses questions such as: What about the ability of the existing systems to meet the expectations of the domain experts? and What elements run counter to these expectations? Synthesis of these findings leads to component design strategies and hypotheses. In our scenario, end-users were using a primitive DOS graphics-based system. A thorough study of the existing system and user opinion helped DAKOTA to become a robust problem-solving application.

The qualitative assessment of the learnability and usability are next tested with the domain experts interacting with varying levels of simulated systems. In this iterative design process domain, experts are provided with various choices of placement of the GUI object (e.g., buttons, menus, etc.), and their feedback is applied to the simulated systems. Experts are then given a composite task from their application domain, which further helped to refine the usability of the system. This type of iterative process should be continued even beyond the point of the analysis phase and carried well into the design and initial implementation phases.

Knowledge should be built into the system to monitor the usage of the on-line help component. This information includes how many times a particular help topic/section/section is visited and how many different domain users have visited them. This is helpful in upgrading the system both in terms of usability and in continually enhancing the system [17]. For example, if a particular on-line help section/topic is visited by many domain users, then there is a possible problem area in usability or in the way the process is being modeled in the interface. With this assistance, the problem can be located and corrected after discussions with the schedulers. The on-line hypertext help sub-system in DAKOTA has this feature built-in; it stores information about how many times a particular topic is referred to in the repository.

3.1.10 Color

Use of color in human-computer interaction provides an opportunity for the interface designer to effectively communicate the dynamics of the system. Color assignments to different components of the display objects are part of the design activity [13]. Apart from the interface designer codifying colors of different display components, the designer should also provide the end-user the option of choosing a color of choice for some display objects in a controlled manner. But research has shown, when users are given free access to color, a wide variations of color combinations are produced, destroying the aesthetics of the presentation module of the system [14] [18]. Thus, there should be limited flexibility for changing the color of display items.

The schedule screen in DAKOTA uses colors to differentiate soft time and hard time violations. Soft time is a time constraint on the request leg that is preferable but could be violated; hard time should never be violated. If the user, while scheduling a request, violates a hard time constraint, that request leg turns red, thus sending a strong warning to the user about the violation. Soft time violations are shown in yellow and the request legs that are bound to all time constraints are shown in green. Colors are also used on the map screen to differentiate different missions, which aids the user in possibly merging two or more missions, provided time constraints are not violated.

3.1.11 Focus

A good user interface allows the user to clearly focus on the material with which he is working. Thus, the user should not be forced to focus on any functions other than the one intended, or no focus at all should be provided. In designing a user interface, the user of the system should not feel the need to only follow a path provided by the system. The user should be able to make changes with the freedom of undoing the work later if necessary. And the order of the actions performed by the user should not be constrained wherever possible, provided application integrity is not affected. There are three ways that users can be allowed to interact with the system:

1. Freely allowing the user to change instances of the data. Just before passing the data to the back-end or the application domain, perform the necessary data integrity check, and leave the rest to be handled by the application domain.

2. Perform a data validation and integrity check after every user interaction with the system. Also provide only the necessary subject and object to the user for interaction. This method advocates full control over the way users interact with the system.

3. Use a mix of both of the above methods, thereby providing some freedom to the user and controlling the way user interact with the system.

The request screen is used to enter travel requests submitted by the clients. This screen is divided into four sections. The first section that is at the top of the request screen is used to capture request header data. This includes the name of the person initiating the request, the contact phone number, date on which the request is opened, etc. The second section is used to capture all the information about all the passenger's flight request. The third section captures the origin-destination pair for each request leg. Finally, there is a remarks section used to put remarks related with the request if any, this remarks field could be used to tell the scheduler of any particular constraints or preferences by the passengers at the time of scheduling. Within the second and third sections, there is a full-fledged editing facility. The data entry person can freely add, modify, and delete items within these two sections. This capability gives the user freedom in making a mistake and coming back to correct it later without imposing the order in which data are entered by the user interface. This capability falls into the first of the above-listed methods, where the user is freely allowed to interact with the user interface, and very little restriction is posed in the way action is taken with various display objects.

In another scenario, the map screen shows one day's fleet activity. The map screen (refer to the appendix for more detail) contains various screen objects with which a user can directly interact. Initially, when the screen shows up on the display device, only the text field for entering the date is given focus. All the rest of the objects are grayed out or desensitized. This concept applies the second of the above-listed methods, where application takes total control of the way in which the user interacts. The reason for obtaining total focus of the user to a text data entry field is that, without a valid date, the rest of the display objects on the map screen will not make any sense.

3.1.12 Modes

This section discusses modes, or hidden information. Modes refer to variable information affecting the meaning of what the user sees and does [15]. For example, a novice user of the 'vi' editor can easily become confused between the edit and command mode. The command mode allows the user to enter commands to navigate around the text file, and the edit mode allows the text file to be changed. The reason for the confusion is either that the mode is not made evidently visible on the screen, or it is not noticed by the user. However, there are also some advantages in using modes:

1. Simplifies program coding and testing, as the number of combinations is reduced in a particular mode. If there is only one mode, it would have to be tested for combinations of the combinations in all the different modes.

2. Protects the user from mistakes. For example, a command "destroys something" does not have any effect in the standard mode, other than causing a warning; a separate mode must be entered before "destroy" is activated.

3. Makes invisible or desensitize a few screen objects in certain modes, thereby protecting the user from making those choices that are not appropriate to a particular context.

4. Provides a means for an application system to take control over user interactions and guide them through the process.

In DAKOTA, modes are used in places where it is appropriate to enforce a logical sequence of steps in the way the user interacts with the system. This method also prohibits users from unnecessarily executing actions that are not relevant to the scenario. Appendix Figure A.1 shows an atlas data entry screen used to add a new atlas record, modify an existing atlas record, and delete the ones not used from the database. These three operations are not related to one another in any way. They can be performed in a totally disjoint manner. Therefore, three modes exist in this screen, called the add mode, modify mode and delete mode. The text field on the top right corner of the screen displays the mode the user is in at any point of time. In one mode, only certain things are allowed. For example, in delete mode, the atlas data will not be able to be modified on the screen. Similarly in modify mode, the system will not allow the user to delete the atlas record. This feature protects the user from accidentally removing any data from the database. A particular mode can be initiated by selecting either ADD/MODIFY/DELETE from the option menu button. Figure A.1, shows NO EDIT in the option menu, clicking on the option menu will drop down other options that can be selected. The user can return from a mode either accepting or discarding the operation by selecting the OK button or CANCEL button, respectively. Therefore, the user is provided with only one way to enter and leave a mode, thereby avoiding all the confusions that could be created. Also, all the display objects that are not relevant to the context are grayed out or desensitized. Thus, the user will not have access to display objects that the system does not want accessed.

4. System Operations and Functions

Section 2 introduced the scheduling problem and provided an overview of system behavior. This section presents the functions of the system and how DAKOTA helps in solving problems. First is an explanation of how the flying travel request is entered into the system, the starting point for the scheduling process. Next, a description is given of how a single request can be manually scheduled followed by scheduling more than one request in a session. Finally, this section explains how manual and automatic scheduling using an optimization model are used together in scheduling a group of requests. This section lists the steps involved in a scheduling process. An example problem and how it is solved is given in Appendix-E.

4.1 Request Entry Screen

The request entry screen is used to enter all information pertaining to a request. Entering request information is performed either by the client who wants to fly or by schedulers on behalf of the clients. All the information is validated before it is stored in the database. The type of validation performed on input data depends on the type of data. More details about validation are given in subsequent sections of this section. There are four main parts to the scheduling screen (Figure 4.1 shows a sample request screen):

- Request Header
- Request Passenger Details
- Request Leg Details
- Request Remarks

4.1.1 Request Header

The request header section is used to input the requesting date, the requesting person contact phone number, and the aircraft type preference, if any. All the information in this section is optional except the requesting date. The entered date should be a valid date; and if the user enters an aircraft type, it is checked against the aircraft database to ensure that it is a valid aircraft type.

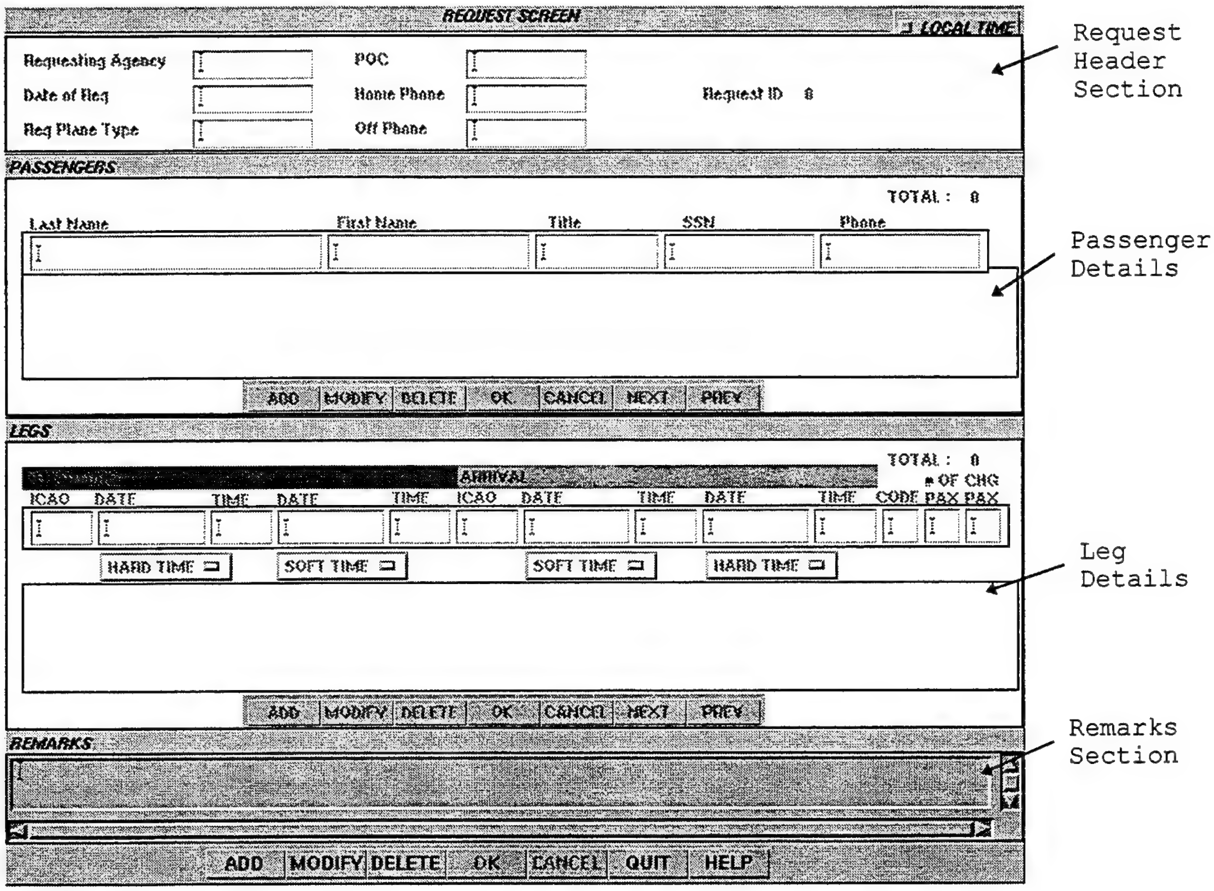

Figure 4.1 Request entry screen.

4.1.2 Request Passenger Details

Passengers traveling in a request are entered in the passenger detail section. The request screen shows the passenger's full name, social security number, and phone number. Details about five passengers are displayed at the same time; if more than five exist in a travel request the user can browse through the list of passengers using the NEXT and PREVious button. This screen is used to add or remove passengers in a travel request. Modification to passenger details if any, can be done in the passenger data entry screen. Clicking on a passenger will highlight the row and make that row current; pressing the delete button will delete the highlighted passenger from the request. Pressing the ADD button creates a blank row following the highlighted row, and the cursor moves to the editable text field. While adding a passenger to a request the user must input only the passenger's first name, last name, both names or some parts of the name and the system will retrieve other information and display it on the screen. If there is more than one passenger who satisfies a particular name a dialog box pops up with the list of all names and social security numbers of the passengers. The user can select the passenger they want to be on the request by using the mouse device. For example, if only A is entered into the last name field, the system retrieves all the passengers whose last name starts with A and displays them in a dialog box. One name could be chosen from the list of names in this request. Therefore, the whole name of a passenger need not be remembered. A passenger is added to the list of passengers traveling

on that request once the user presses the save button.

4.1.3 Request Leg Details

There are two parts to a request leg entry: departure-arrival data entry and passenger to request leg association. Departure-arrival entry is done in the request leg section of the request screen, and passenger to request leg association is done in a separate dialog box. A request leg is a combination of departure and arrival, location and time information. Every location is identified by a 4-character length identifier, also called ICAO. Departure and arrival time are expressed as local date and time. In addition to the time, the user must also provide the firmness of the time, hard or soft. Hard times are those that cannot be violated, and soft times are preferred times which can be altered if necessary to obtain a better schedule. To summarize, a request leg constitutes the following information:

- Departure ICAO
- Earliest departure date/time
- Latest departure date/time
- Indication of firmness of departure time
- Arrival ICAO
- Earliest arrival date/time
- Latest arrival date/time
- Indication of firmness of arrival time
- Priority code
- Number of passengers in the request leg

There are two time windows in a request leg. The earliest and latest departure date and time in the above list refer to the departure time window, and earliest and latest arrival date and time refer to the arrival time window. If the user inputs a departure or arrival time, the system will automatically prompt other times based on the speed of the fastest and slowest aircraft that can fly between two stops or locations. The user can either change the time or accept whatever the system provides. For example, if the departure and arrival locations are EDAR and EDDN, respectively and the earliest departure time is known to be 4/NOV/94 01:00, the system calculates other times which are the earliest time the fastest aircraft in the fleet will arrive at EDDN is 4/NOV/94 01:30 and the latest time the slowest aircraft in the fleet will arrive at EDDN is 4/NOV/94 04:00. This is just a suggestion; the users can either accept these suggested times or change at will to suit their needs.

The second part of this section is associating a passenger to a leg. This is done in a separate dialog box which displays if at least one of following conditions is satisfied:

1. The total number of passengers in the request is not equal to the number of passengers traveling in a request leg.

2. The number of passengers in the current request leg is not equal to the number of passengers in the previous request leg.

3. The users has typed 'Y' in the CHG PAX field, meaning that he wishes to change the passenger association to a request leg.

Figure 4.2 shows a sample passenger-leg association dialog box. There are two list boxes, one on the left-hand side containing the list of passengers who are on-board in the request leg and one on the right-hand side contains the list of passengers who are not traveling in that request leg. The user can move passengers between on-board and off-board lists by clicking on an item.

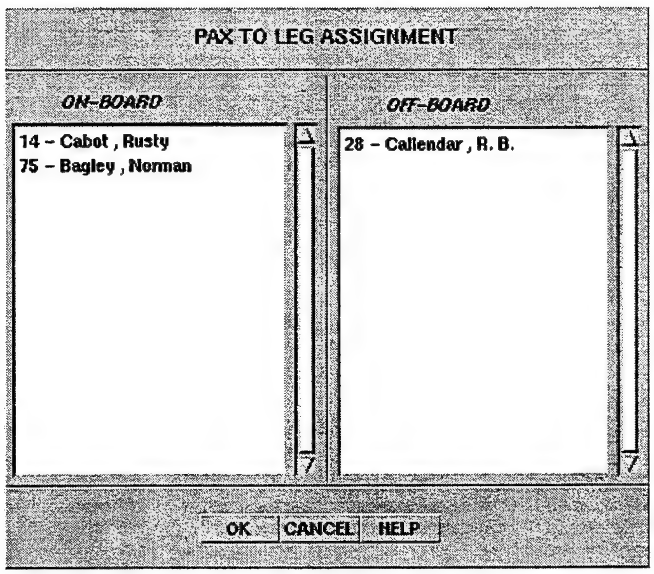

Figure 4.2 Passenger to leg association dialog box.

A validation routine checks to see if the entered date and time are valid. It also checks if the entered departure and arrival locations are in the atlas database. DAKOTA provides a great deal of flexibility to end-users by letting them enter all or part of the location identifier or ICAO. If user enters part of the ICAO code, the system retrieves all locations that match the criteria and displays them in a dialog box. The user can select the location they want by clicking on an item. If the selection criterion matches only one location in the database, it retrieves that ICAO code and fills it in the location field on the screen.

The user can see five request legs at a time. The user can browse through all the request legs using the NEXT and PREVious button. The total number of request legs in a request is shown on the right top corner of the request leg section, as shown in Figure 4.1. The user can add a new request leg, modify an existing request leg or delete a request leg by pressing the add, modify, or delete button, respectively. Clicking on a leg will highlight and select a request leg for modification or deletion. If the user presses the modify button, the highlighted request leg will appear in the editable text field where the information can be changed. Pressing SAVE or the CANCEL button will either store changes in the database or discard them, respectively. Similarly, pressing the DELETE button will mark a highlighted request leg for deletion. Once the user presses the save button, all request legs marked for deletion are removed, both from the database and from the list shown on the screen. Now any request

leg below the deleted request leg shifts up to fill in the deleted leg's spot.

4.1.4 Request Remarks

The remarks section is used to enter any comments that would help the scheduler during mission scheduling. It is optional; the user could either choose to enter some text or leave it blank. There is no validation performed in this field, but it is stored in the database.

The above descriptions indicate the user can input request header information; add and remove a passenger in a request; add, modify, and delete a request leg; make a passenger go on-board or off-board a request leg; and finally enter request remarks. All of these operations pertain to manipulating a request. The user can retrieve a request and all its passenger and leg information from the database by providing one or more the of following items:

- The point of contact (POC)
- The request date
- The request identifier

Once the system identifies a request, it retrieves all the data and displays it on the screen. If more than one request satisfies the selection criteria, a dialog box pops up with all the requests. The user can choose the request they want to work with by clicking on a line in the dialog box. Once a request is selected, changes may be made to that request as mentioned in the previous description.

4.2 Defining and Solving the Problem

The users employ the scheduling screen to schedule the requests for travel. There are two distinctive steps in a scheduling session, defining the problem and solving the problem. Output from a scheduling session is one or more mission(s) with a scheduled request leg(s). A scheduling screen can display only one scheduling session, but a scheduler can work on more than one scheduling session at a time by having more than one scheduling screen open.

The definition of a problem consists of identifying unscheduled request legs, then identifying all the available aircraft. Aircraft and tail number are used interchangeably. The tail number is a 5-digit number that uniquely identifies an aircraft. The next step in the scheduling process is solving the problem, which can be done in three ways:

- Using only the manual scheduling process
- Using only the automatic (optimiztion) solver
- Using both the manual and the automatic techniques

The output of this process is one or more mission(s) with the aircraft marked as being used for a specific block of time and one or more unscheduled request legs marked as being scheduled. In the subsequent sections, a description is given of each of the above solving techniques is used to achieve an optimum schedule. Before going into actual scheduling, the

next section presents an overview of the scheduling screen.

4.2.1 Overview of the Scheduling Screen

There are two parts to a scheduling screen. The mission work area is on the top half of the scheduling screen, and the request work area is on the bottom half of the scheduling screen. There are several buttons associated with the mission work area and a few the with request work area. These buttons are used to manipulate the missions and requests displayed in their respective work areas. The mission work area shows information about missions. Details about one mission at a time are displayed. The NEXT and PREVious buttons are used to browse through other missions in a scheduling session. The first line in the mission work area displays mission header information including mission identifier, tail number, mission type, SPAR number (it is a number specific to USAFE for identifying a mission), and finally, the status of the mission. Data from the second line onward pertain to legs in that mission, also called mission legs. A mission leg consists of the following:

- Departure location and time
- Arrival location and time
- ETE (Estimated Time Enroute)
- Priority of the mission

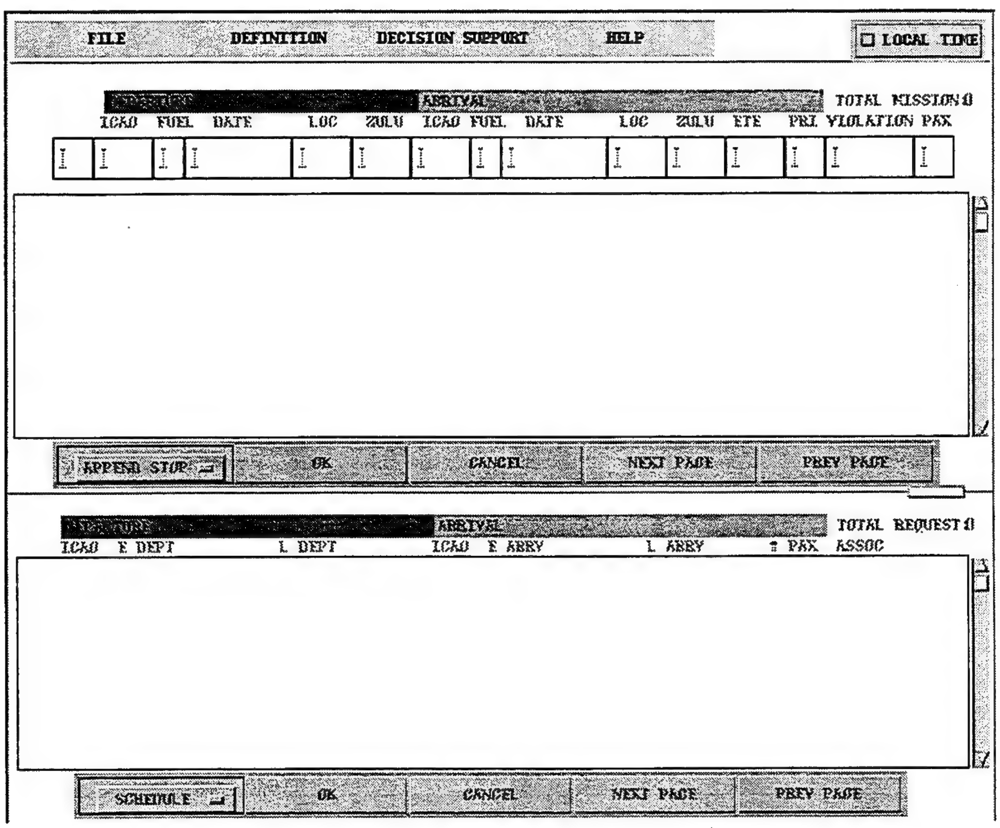

Figure 4.3 Scheduling screen.

The mission work area displays seven mission legs at a time. More mission legs, if there are any, can be seen by scrolling, using the scrollbar located at the right side of the mission work area. There is an array of editable fields just above the mission work area. These editable fields are used to input and change information about a mission leg. The subsequent paragraphs outline how to input and change mission leg information. The total number of missions in the current scheduling session is shown above the mission work area.

The request work area shows information about one request at a time. The NEXT and PREVious buttons are used to browse through other requests in the current scheduling session. The first line in the request work area displays request header information. This line contains a request identifier, point of contact, and name of the leading passenger. Data from the second line onward pertains to request legs including requested departure and arrival location and time and total number of passengers in a request leg. The request work area displays seven request legs at a time. More request legs, if any, can be seen by scrolling, using the scrollbar located at the right side of the request work area.

The functions of the buttons in the mission and request work area are described in the manual scheduling section. In addition, there are three pulldown menus located at the top of the scheduling screen. They are used to manipulate a scheduling session, define a problem, and invoke the automatic scheduler. Figure 4.3 depicts a scheduling screen with sample data. The next section discusses how to define a problem.

4.2.2 Defining a Problem

Defining a problem is the first step in a scheduling process. The steps below are followed in defining a problem:

- Identify aircraft for utilization.
- Identify unscheduled requests or request legs.

4.2.2.1 Identifying Unscheduled Requests or Request Legs.

Identifying is completed by clicking on the SELECT REQUEST button from the DEFINITION pulldown menu located at the top of the scheduling screen. Figure 4.4 shows a sample SELECT REQUEST dialog box. The user could select a request for scheduling by entering either request ID, name of the lead passenger, point of contact, or any combination. The system retrieves all the unscheduled requests from the database that satisfy the criteria given in the dialog box and displays in the list box. The user can continue selecting requests and accumulate them with already retrieved requests. Double-clicking on an item in the list box will remove the request leg from consideration for scheduling. Once the user is satisfied with the list of unscheduled request legs for scheduling, he or she can press the OK button. This procedure will display relevant information about all the selected request legs on the bottom half of the scheduling screen.

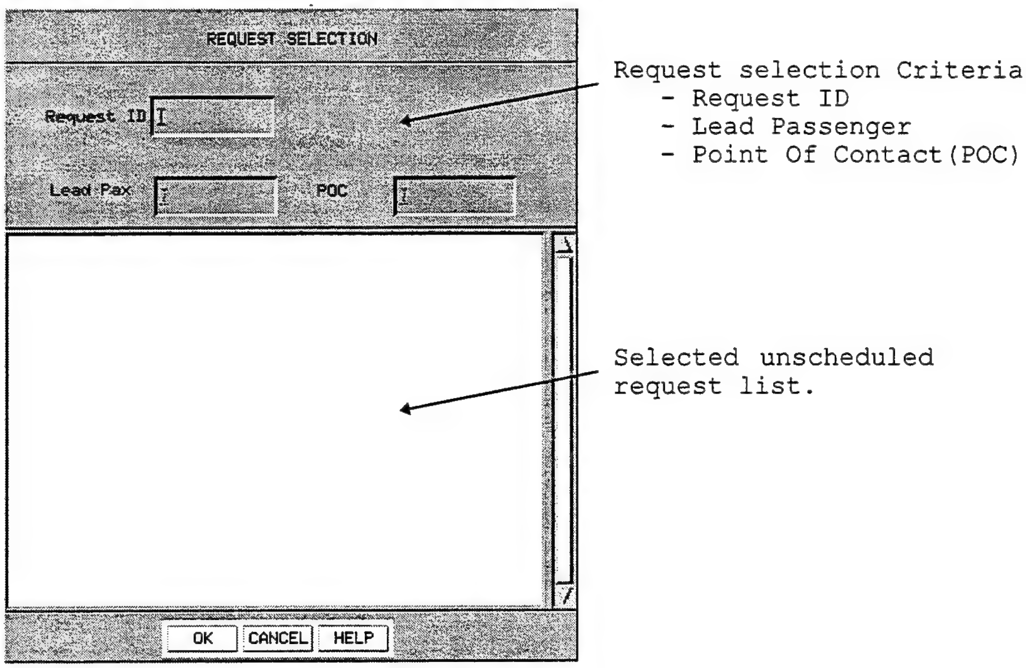

Figure 4.4 Unscheduled request selection dialog box.

Appropriate selection of unscheduled request leg(s) is essential factor in obtaining good optimum schedule. Tools like the map screen can help make good decisions about a request selection. Basically, the map screen allows the user to view all the scheduled missions and unscheduled request on a daily basis. This screen also shows statistics about aircraft utilization in terms of total hours flown against annual budgeted hours, and a time window of all aircraft utilization and the one available for scheduling on a given day. This information is useful in identifying aircraft for scheduling. More about the map screen and how it aids in visualizing a day fleet activity is narrated in Section 6.

4.2.2.2 Identifying Aircraft.

A resource consists of ordered pairs of aircraft and operational time windows. An operational time excludes the amount of time an aircraft is in service, in maintenance, specially allocated, or unavailable for scheduling. A single aircraft in a day with a disjoint operational time window can be considered as a unique resource. The scheduler must keep in mind the approximate time windows of unscheduled request legs selected for scheduling in the previous step. This will help one to choose an appropriate resource for optimum scheduling. Identifying a resource takes place in one of following ways:

- Identifying existing missions flown by aircraft which can accommodate unscheduled requests
- Selecting missions and modifying aircraft available time (i.e., increase the time window of aircraft utilization for selected missions which enables scheduling unscheduled request legs)
- Creating new missions using an available aircraft by blocking a specific span of that

aircraft's time.

The first step above is implemented by clicking the SELECT MISSION button in the DEFINITION pulldown menu at the top of the scheduling screen. Figure 4.5 shows a sample SELECT MISSION dialog box. A mission is selected by entering either the existing mission ID or mission date. DAKOTA retrieves all the missions that satisfy the criteria and displays them in the list box. Selection of a mission triggers selection of all scheduled request legs in that mission. Double-clicking on a mission removes that mission from the scheduling session. The user can continue selecting more missions and append them to the list of missions already selected. Once satisfied with the list of missions, the user presses the OK button to display selected missions in the mission work area of the scheduling screen.

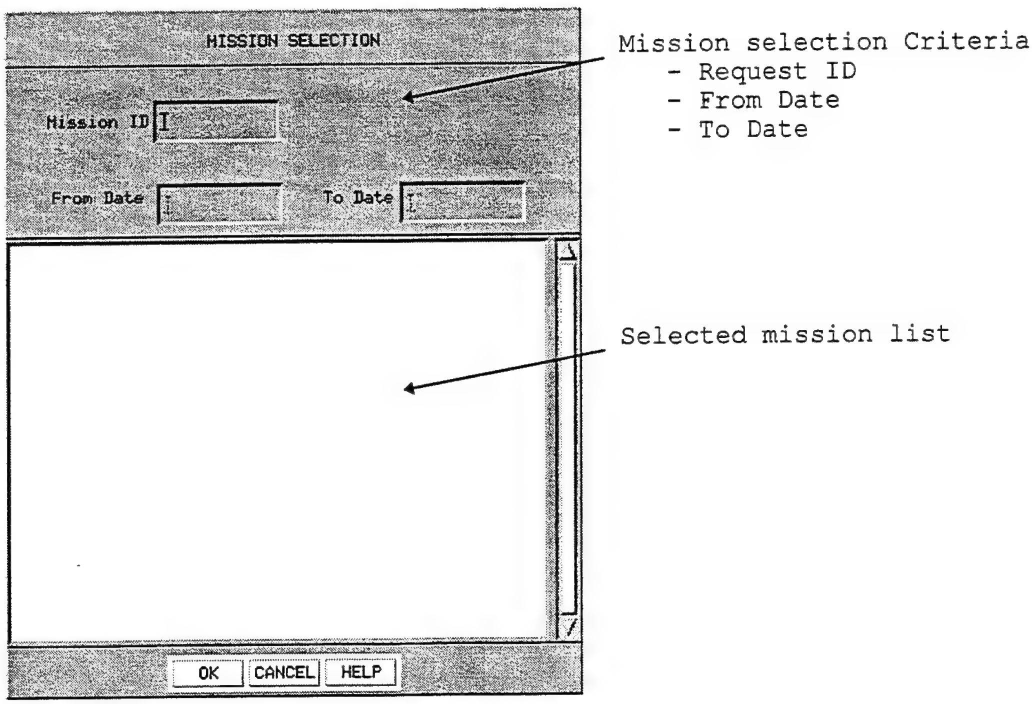

Figure 4.5 Mission selection dialog box.

In the second method, the time window of the mission is modified such that it accommodates unscheduled request legs selected in the previous step of problem definition. Clicking on the MODIFY MISSION button under the DEFINITION pulldown menu displays a dialog box. Figure 4.6 shows a sample MODIFY MISSION dialog box. For this method to work, the user has to previously have selected the mission using the first method. The earliest time and latest time fields in the dialog box are changed to accommodate an unscheduled request leg.

The third method is creating a new mission with an available aircraft and time window. Clicking on the NEW MISSION button under the DEFINITION pulldown menu will pop up a dialog box similar to the one in Figure 4.3. The user has to enter all the fields in the dialog box and press the OK button. This will save the mission in a temporary buffer after validating the aircraft type, tail number, and time window. A time window for aircraft utilization is the time between the earliest and latest time. This time window should not overlap with any mission already in the system. Now we have defined input for the

scheduling process. In the next paragraph, a description is given of how these data are used in scheduling unscheduled request legs.

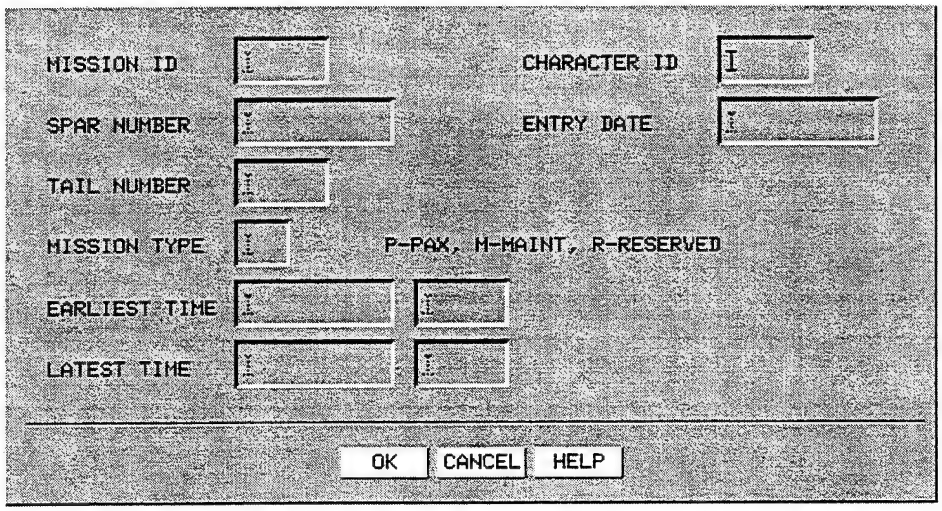

Figure 4.6 Mission header data entry screen.

4.2.3 Manual Scheduling

Before presenting scheduling as such, an overview of the operations available in the scheduling screen is presented. Then the different types of scheduling problems faced by schedulers are listed, including how the operations can be used to generate effective schedules. There are two sets of operations available; one for manipulating missions and another for manipulating requests. The operations are as follows:

Mission related:

- Append Stop
- Insert Stop
- Modify Leg
- Remove Origination Stop
- Remove Destination Stop
- No Edit
- Adjust Forward
- Adjust Backward
- Next Mission

- Previous Mission
- Save Mission
- Cancel Mission

Request related:

- Unschedule Request Leg
- Schedule Request Leg
- Deselect Leg
- Next Request
- Previous Request
- Save Request
- Cancel Request

4.2.3.1 Mission-Related Operations

Append Stop: This function is used to append a stop to a current mission leg. Clicking on a mission leg will make it current and highlight that row. After making a leg current, select the APPEND STOP button from the option menu. This selection will create a new row after the current row and let the user into the input origin location and time in the editable text field for the new row. Departure details for the new row are derived from the current row. After entering all information in the editable text field, press the OK button to validate and save changes to the database. In other words, this operation adds a stop after the destination stop of the current row. Table 4.1 shows before and after effects of appending a stop.

Before appending a stop(EPWA) to the first row:

	Miss	Origin				Destination			
No.	Id			Date	Time			Date	Time
1	30500	EDAR	Y	5/Apr/95	0830	EDBT	N	5/Apr/95	0915
2		EDBT	N	5/Apr/95	1030	EDAR	N	5/Apr/95	1115

After appending a stop(EPWA) to the first row:

	Miss	Origin				Destination			
No.	Id			Date	Time			Date	Time
1	30500	EDAR	Y	5/Apr/95	0830	EDBT	N	5/Apr/95	0915
2		EDBT	N	5/Apr/95	1030	EPWA	N	5/Apr/95	1115
3		EPWA	N	5/Apr/95	1200	EDAR	N	5/Apr/95	1325

Table 4.1 Effect of appending a stop to an existing list of mission legs

Before inserting a stop(EPWA) to the first row:

	Miss	Origin				Destination			
No.	Id			Date	Time			Date	Time
1	30500	EDAR	Y	5/Apr/95	0830	EDBT	N	5/Apr/95	0915
2		EDBT	N	5/Apr/95	1030	EDAR	N	5/Apr/95	1115

After inserting a stop(EPWA) to the first row:

	Miss	Origin				Destination			
No.	Id			Date	Time			Date	Time
1	30500	EPWA	Y	5/Apr/95	0700	EDAR	N	5/Apr/95	0745
2		EDAR	Y	5/Apr/95	0830	EDBT	N	5/Apr/95	0915
3		EDBT	N	5/Apr/95	1030	EDAR	N	5/Apr/95	1115

Table 4.2 Effect of inserting a stop to an existing list of mission legs

Insert Stop: This function is used to insert a stop to a mission leg. Clicking on a mission leg will make it current and highlight that row. After making a leg current, select the INSERT STOP button from the option menu. This will insert a new row above the selected row and let the user enter destination location and time in the editable text field. After entering all information, press the OK button to save changes to the database. In other words, this operation inserts a stop before the origin stop of the current row. Table 4.2 shows before and after effects of inserting a stop in a mission leg.

Modify Leg: This feature is used to make changes to the current mission leg. Clicking on a mission leg will make a mission leg current and shows that row in highlighted color. After making a leg current, press the MODIFY LEG button from the option menu. This will bring current mission leg-related data to the editable text field. The user can change information and press the OK button to save changes.

Remove Origination Stop: This feature is used to remove a departure or origination location from a mission leg. First, make a row current by clicking on a mission leg. Select REM-O-STOP from the option menu. This will remove the departure information from a mission leg and assign current mission leg's arrival information to the previous mission leg, if one exists. Otherwise the whole mission leg is removed.

Remove Destination Stop: This option is used to remove arrival or destination location from a mission leg. First, make a row current by clicking on a mission leg. Then, select REM-D-STOP from the option menu. This selection will remove the arrival information from the current mission leg and assign the next mission leg's arrival information to the current mission, if one exists. Otherwise, the whole mission leg is removed.

No Edit: This option is used to browse through the list of missions in a scheduling session. Browsing the mission list is done using the NEXT and PREVious buttons.

Adjust Forward and Adjust Backward: This feature is provided as a convenience for the

scheduler. As a result of editing the times of mission legs, there may be turnaround violations. Turnaround time is the minimum time necessary for aircraft refueling. This feature is described using an example in Table 4.3.

Before editing mission leg's time:

	Departure	Arrival
Mission leg 1	8:00 AM	11:00 AM
Mission leg 2	12:30 PM	3:00 PM

After editing mission leg's time:

	Departure	Arrival
Mission Leg 1	9:30 AM	12:30 PM
Mission Leg 2	12:00 PM	3:00 PM

Table 4.3 Before using ADJUST FORWARD or BACKWARD feature

The departure time of mission legs violates the minimum turnaround time of 45 minutes. In fact, it is earlier than the arrival time of the previous leg. Adjust Forward would shift the time starting from the first mission leg. Adjust Backward would shift the time starting from the last leg. The effect on the mission legs after choosing either adjust forward or backward is given in Table 4.4:

Adjust forward:

	Departure	Arrival
Mission Leg 1	9:30 AM	12:30 PM
Mission Leg 2	1:15 PM	4:15 PM

Adjust backward:

	Departure	Arrival
Mission Leg 1	8:15 AM	11:15 AM
Mission Leg 2	12:00 PM	3:00 PM

Table 4.4 After using ADJUST FORWARD and BACKWARD feature on

the edited mission leg times

Adjust Forward slides times forward as needed to ensure that the minimum turnaround time is not violated. The departure time of the first leg remains unchanged. Adjust backwards does the same, except it slides times backwards, while the arrival time of the last leg remains the same and all other times are changed to accommodate turnaround. This sliding of times does not affect the actual flight time.

Next Mission: This button is enabled when there is a next mission to the current session. Pressing the NEXT button will clear mission work area and displays the next mission header and all its mission legs.

Previous Mission: This button will be enabled when there is a previous mission to current mission. Pressing the PREV button will clear the mission work area and displays the previous mission header and all its mission legs.

Save Mission: At any point of time there are at least two copies of all the missions and request information preserved in main-memory. One copy will have original data as retrieved from the database. The second copy is original data with user changes. This copy gives the user an option of reverting to original copy if the user is not happy with the changes made. Pressing the OK button after any change will perform a set of validations and updates to the second copy. The second copy of missions and requests is updated only if user data passes validation. The list of validations performed is given in Section 5.

Cancel Mission: Pressing the CANCEL button discards all user changes and will not make any change to any of the items mentioned above.

4.2.3.2 Request-Related Operations

Schedule Request Leg: This feature is used to schedule an unscheduled request leg on one or more mission legs. The user has to click on the SCHEDULE LEG button, followed by clicking on the leg the user wants to schedule. The system will automatically highlight the mission legs on which the unscheduled request leg will fit. If there is more than one place that the request leg will fit, clicking the NEXT and PREVious button in the request work area will help go through the alternatives. The user can click the OK button if satisfied with an alternative mission leg(s) shown in the mission work area. This will schedule an unscheduled request leg to highlighted mission leg(s). It will also increment the number of passengers flying in that mission leg with the number of passengers in the selected unscheduled request leg. This scenario is explained with the following example.

Let us assume there are three mission legs and one unscheduled request leg, as in Table 4.5. To schedule Request Leg 1, clicking on Request Leg 1 in the request work area will highlight contiguous mission legs that fly from request leg origin to destination. Request Leg 1 goes from locations A to F; Mission Legs 1, 2, and 3 together go from locations A to F. DAKOTA will determine all the combinations of contiguous mission legs that match a request leg and display them to the user. The system's suggestion can be accepted by pressing the OK button when one of the alternatives is shown in the mission work area. This will schedule that request leg onto the highlighted mission leg(s). The system will display a warning dialog box, if any violation occurs.

Mission legs in mission work area

	Departure ICAO	Arrival ICAO
Mission leg 1	A	B

Mission leg 2	C	D	
Mission leg 3	E	F	
Request leg 1	A	F	Request legs in request work area

Table 4.5 Scheduling a request leg to mission leg(s)

Unschedule Request Leg: This option is used to unschedule an already scheduled request leg. First, click on the UNSCHEDULE LEG button from the option menu; then, select a scheduled request leg from the request work area. This will unschedule the selected request leg and will also decrement the number of passengers flying in the mission leg(s) supporting that selected scheduled request leg.

Deselect Leg: This option allows the user to remove a request leg from a scheduling session. Press DESELECT LEG from the option menu, and select all request legs that need to be removed from the current scheduling session. Finally, press the OK button to accept the changes.

Next Request: This button will be enabled when there is a next request to current request. Pressing the NEXT button will clear the request work area and display the next request header and all its request legs.

Previous Request: This button will be enabled when there is a previous request to current request. Pressing the PREV button will clear the request work area and display the previous request header and all its request legs.

4.2.3.3 Steps in Manual Scheduling

Typically, schedulers will be faced with one of the following scheduling scenarios.

- A single request is scheduled with single mission.
- A single request is scheduled with many missions.
- Many requests are scheduled with a single mission.
- Many requests are scheduled with many missions.

The first of the above is one of the simplest problems to solve. The successive scenarios become more difficult as the number of requests and missions increase in a single scheduling session. The sample scheduling problem in Section 2.2.2 covers all the different type of scheduling scenarios. The steps below summarize the operations that are involved in a typical scheduling.

1. Select unscheduled request legs, as described in Section 4.2.2.1.
2. Create a new mission, or select an existing mission, as described in Section 4.2.2.2.
3. Create mission legs corresponding to each request leg, without violating the endurance of aircraft. If a request leg takes more time than the endurance of the aircraft, insert more stops for aircraft refueling. These stops are called fuel stops. Stops can be added to a mission using either INSERT STOP or APPEND STOP. Sometimes it is necessary to modify the time window or departure or arrival ICAO to meet some of the request

time and location constraints. This can be accomplished by using MODIFY STOP.

4. Associate unscheduled request legs with mission legs, as described in Section 4.2.3.1.

5. Look for any violations, and remove them if possible. Severe violations are shown in red. The next paragraph discusses more about different types of violations and how DAKOTA identifies them.

6. Save the work in the database as described in Appendix B.

The next paragraph describes the different types of violations that can occur in scheduling.

4.2.3.4 Different Types of Violations

There are five types of violations possible in a scheduling scenario:

- Endurance violation
- Turnaround time violation
- Capacity violation
- Hard time violation
- Soft time violation

Hard and soft violations are related to requests, and the rest are associated with aircraft and missions. All violations, except soft time, are considered severe and must be resolved. Severe violations are shown in red, soft time violations are shown in yellow. Request and mission legs are shown in green, whenever there are no violations. Endurance violations occurs when a scheduled flight time is more than the endurance of the aircraft. For example, the endurance of aircraft C-21 is 4 hours. If a mission leg is created with flying time of more than 4 hours, the endurance column in the mission work area will show in red, indicating the endurance of the flight is exceeded.

Regarding a turnaround time violation, there must be a 45 minute ground time between every mission leg. This is to accommodate the time spent for landing, takeoff, and other ground activities. Turnaround time violations occur when a mission leg is scheduled without allowing a minimum of 45 minute ground time. Whenever this violation occurs, the origin mission leg will display in red. A capacity violation happens when the capacity of the aircraft is exceeded (i.e., scheduling more passengers on a flight than could be accommodated). Whenever this occurs, the PAX column in the mission work area shows in red. A hard time violation is associated with a request. It happens when the hard time constraint of the request leg is violated. A soft time violation is also associated with a request. It happens when the soft time constraint in one of the request legs is violated. It is not a severe violation, so it is shown in yellow. Other violations are shown in red.

4.2.4 Automated Scheduling

A heuristic algorithm [5] was developed to support a complete decision support tool for automated scheduling. DAKOTA integrates manual and automated scheduling to provide the user with the best of both worlds. Steps involved in automated scheduling are as follows:

1. Define the problem, as described in section 4.2.2.

2. Invoke the heuristic algorithm by selecting the DECISION_SUPPORT->RUN button from the pulldown menu in the scheduling screen. The set partitioning algorithm tries to schedule all unscheduled requests to missions. The output of the algorithm is a sequence of scheduled request legs on to mission legs. The user can view them in the scheduling screen with scheduled requests and its corresponding mission in the request and mission work area, respectively.

4.2.5 Combination of Manual and Automatic Scheduling

Both manual and automated scheduling work seamlessly. Automated and manual scheduling can be performed in any order and any number of times on a given problem. Start the whole process with manual scheduling, making changes to missions manually, and then continue the scheduling process using the algorithm. Later, output from the automated scheduler can either be altered by the human scheduler or accepted as returned from the automatic scheduler. The automated scheduler works better than a human scheduler when there are a large number of requests and missions in a scheduling session. So by using the automated scheduler, the user will be able to schedule the major part of the unscheduled requests. Finally, a schedule that is acceptable to all involved parties can be generated by making minor changes and removing any violations manually to the output from automated scheduler. Once the schedule is in final form, database updates are made to reflect the effects of the schedule on the entire fleet schedule structure.

5. SOFTWARE SYSTEM DESIGN

This section presents the software system design. The system design followed a detailed requirement analysis task. It included discovery of process, refinement, modeling, and specification of the manual process. Typically, a GUI system requires both the developer and the customer to take an active role during the requirement analysis and system specification stage. Whiteside et al.[4] argued that laboratory experiments provide little design guidance. These researchers advocated a field approach in which the specific task and context of the user are of primary importance. In our case, schedulers at the Air Force base in Ramstein, Germany, took an active role in the analysis and design stages. This facilitated the crafting of a software system that effectibely solves the problem of interest.

There are many sub-systems integrated together to form a robust scheduling software system. The heuristic algorithm responsible for automated scheduling is described in [5] the database sub-system tailored for the scheduling paradigm is explained in [6]. In this section, the system design of DAKOTA is described in Section 5.1; the on-line context sensitive hypertext sub-system is described in Section 5.2; the UIL to C compiler is detailed in Section 5.3; finally Section 5.4 will discusses integration strategies of all the sub-systems to form a single state-of-art scheduling system. DAKOTA is a fully data driven software module, we used some of the concepts and ideas in [19] [23] of how to construct and maintain schedule related and spatial data structures.

5.1 DAKOTA Design

A widely accepted principle of application design is that a core application should not rely on a specific user interface. The design then allows the program to run with different interfaces without changes in the core application. With this background, a layered design was created for DAKOTA. Each layer is connected with the other with very little coupling, enabling maintenance and implementing changes to an individual layer with ease. There are a total of four layers in DAKOTA:

- Presentation layer
- Presentation logic
- Data logic
- File interface

5.1.1 Presentation Layer

The presentation layer contains a set of screens. These screens are designed using an UIMS (User Interface Management System) called DEC-VUIT. It is a RAD (Rapid Application Development) tool for quickly prototyping screen interface. Screens generated from the DEC-VUIT tool are saved as text files. These text files contain screen definition in Motif specification language, also called User Interface Language (UIL). The UIL file is later converted to C source file, using the UIL2C convertor described in the subsequent

section. These sets of C source files constitute the presentation layer.

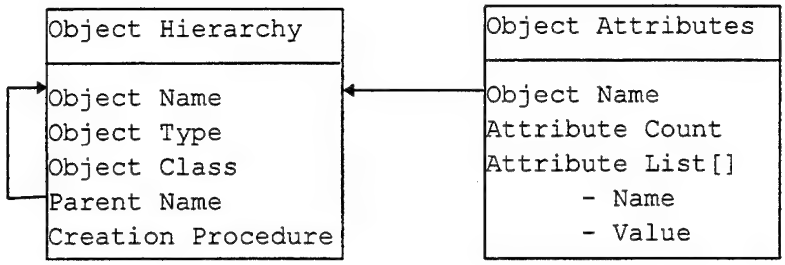

Figure 5.1 Data stores used in presentation layer.

A screen is made up of many display objects. An object could be a widget or control in X windows or the MSwindows environment. The object can also be a combination of more than one widget or control. There are two main data stores, also called data structures, that describe the characteristics of an object and its relationship with other objects. These data stores are referred to as object hierarchy and object attributes. Object attributes data stores describe the characteristics of a display object (e.g., width, height, color, font, position). Object hierarchy data stores describe the interconnection or hierarchy of display objects in a screen. Figure 5.1 depicts the relationship and various members of object hierarchy and attribute data stores. Functions are written to create screens after reading these data stores at run-time. This layer is more static in nature. Any change to the display object is made in the presentation logic layer.

5.1.2 Presentation Logic

The presentation layer responds to events. During object creation, each object is designated to respond to certain events. An event could be generated by a user action (e.g., a mouse click on a push-button, a drag request in a drawing canvas area etc.) or it could be generated by the system (e.g., a screen is losing focus, or the mouse movement has been noticed etc.). Depending on an event, this layer consults with the data logic layer, obtains required data, and modifies the content on the screen, if necessary. This layer also acts as a bridge between front-end screens and the back-end application logic.

5.1.3 Data Logic

The data logic layer maintains data structures used in DAKOTA. There are many data structures, but only the one used in the scheduling screen is discussed here. Figure 5.2 shows the data structures and interconnections between them. There are four main data structures: mission header, mission leg, request header, and request leg. This data structure embeds the following relationships:

- A mission header can have one or more mission leg(s).

- A request header can have one or more request leg(s).
- A mission leg can fly one or more request leg(s).
- A request leg can be accommodated in one or more mission leg(s).

The above relationships cover all USAFE scheduling requirements. There are functions written to add, modify, and remove mission and request header and legs data structures.

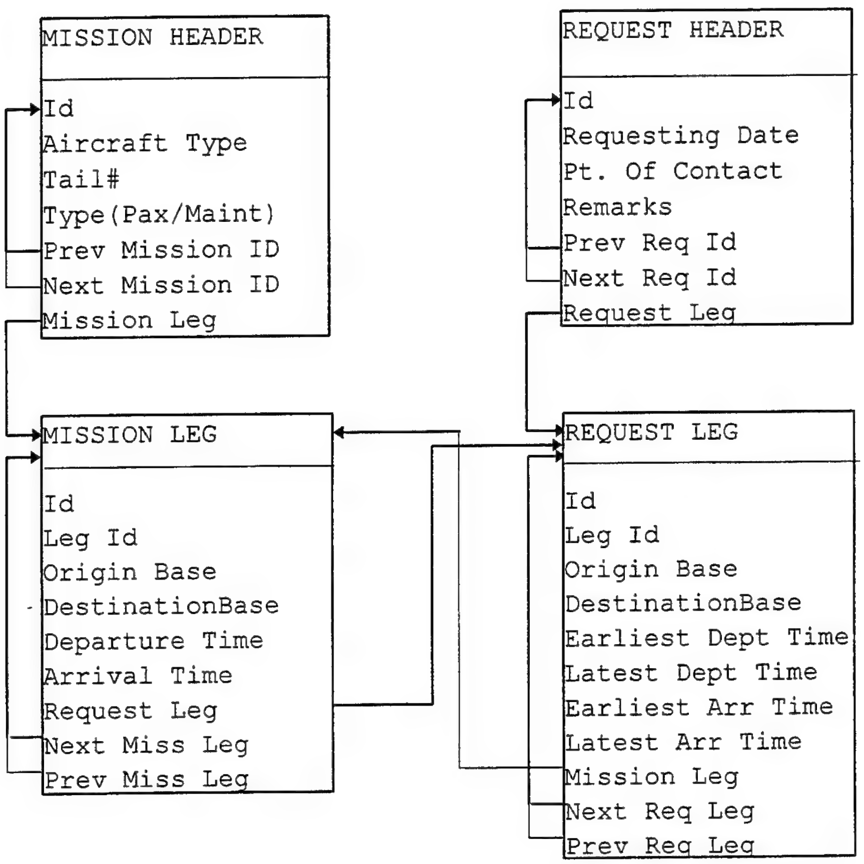

Figure 5.2 Scheduling data structures.

These data structures are created when end-users define the problem as described in Section 4.2.2. Whenever the user initiates one of the operations mentioned in Section 4.2.3, the presentation logic layer captures the user action, validates the user action and call an appropriate procedure in data logic layer. This layer further validates the user action and updates the data structures and its interconnections. The types of change that could happen to these data structures depend on the type of operation selected by the user. For example, if the user clicks on the APPEND LEG button, a mission leg is added after the current mission leg in the mission leg data structure.

Apart from maintaining these data structures, this layer interacts and integrates automatic

scheduling and the manual scheduling process. The automatic scheduler requires data in a different form, so the data logic layer converts the above-mentioned four data structures in a form that can be understood and processed by the automatic scheduler. When the automatic layer completes its processing, all data are transformed to data structures, as shown in Figure 5.2.

5.1.4 File Interface

DAKOTA uses two types of information or data, static and dynamic. Static data remain constant and are used only for display purposes. For example, map screen displays, country boundaries, rivers and lakes are all represented by a sequence of points. Static data are stored in ASCII files. Dynamic information undergoes changes. Dynamic data are stored in DESc [6] database. DAKOTA uses DESc Application Program Interface (API) layer to retrieve, update, and delete dynamic data; static data are handled directly using operating system file open, read, and close system calls.

5.2 Help System

All traditional text, whether in printed form or computer files, is sequential, meaning there is a single linear sequence defining the order in which the text is to be read. Hypertext is non-sequential; there is no single order determining the sequence in which the text is to be read. A few hypertext applications in the market address some of the hypertext functionalities but they are limited. This paper analyzes possible features and functionalities, that a hypertext application could have and an implementation of a sample hypertext application using the X11R5 environment on UNIX. This section lists the advantages and disadvantages of a hypertext system, the features supported in the on-line help sub-system developed as part of the DAKOTA project, and finally, discusses the design of the help system.

5.2.1 Advantages and Disadvantages

What follows are some of the advantages of the hypertext help system:

- Keyword searching is one of the significant advantages of the hypertext system. It allows one to search an on-line document, either by giving full phrases, keywords or part of the them in finding the necessary information.

- Ease of distribution is an advantage due to the system's cost effectiveness in distribution.

- Up-to-date information is provided by changing the help document files in one place instead of making supplement printed copies and distributing the latter method could be expensive, and there is a chance of missing someone when distributing the supplements.

- Linking large libraries is an advantage of the hypertext system while leaving individual parts unharmed.

- Non-linear organization of hypertext documents allows total freedom in organizing documents. With hypertext, the writer layout the document to fit the information, instead of forcing information into an arbitrary structure. In off-line, the document reader has to go to a different page or book whenever encountering cross-references, bibliographic citations, glossary terms, footnotes, etc. In hypertext, just by clicking on the

hot-spot, the reader can go to a different topic temporarily and return back to the original topic.

- Associative thinking is modeled in a hypertext system which more closely resembles the structure of human idea processing by creating a network of nodes and links, allowing for three-dimensional navigation through a body of information. Therefore, hypertext should prove easier to learn, understand, and remember. A sample model is discussed in [21].

- More paths to information - It allows user to move in any direction they feel appropriate in their quest for information. Also an information can be reached in many different paths, if that is relevant to the context.

- Availability of on-line documents is always available when the application system is used as supposed to printed document that can easily get misplaced or lost.

Although, hypertext is advantageous, there also are some disadvantages, as given below:

- The lost in hyperspace phenomenon: The user can easily get lost in vast hierarchies and networks of information. Even if help document is designed properly, there is a chance of one losing track of his initial intentions and getting lost in the volume of information [20].

- Difficult to create and maintain; It requires skill and experience in the problem domain and know-how of end-user expertise. It creates more problem when many people are working in creating and maintaining the help system, regarding who is to create a particular topic and who is to update which links, etc.

- Reading sequences are unpredictable: As hypertext does not impose any order in the reading sequence, so the user can easily get confused if the on-line help document is designed poorly.

5.2.2 Features

What follows is the list of features available in the hypertext help system:

- Hot spots: Certain phrases and images in a help screen are set apart from the rest of the information. These hotspots, or hypertext areas distinguish the on-line help system from the printed off-line documents. They are links to a different topic or help text. When the user clicks on these hotspots, more information is obtained on that hypertext. There are two types of hot spots, hyper links to a new topic and the help pop-up. The first type replaces the current help text with a new help text, the meaning all the text in the help screen is replaced with the text for the new help topic. The second type of hotspots provide a pop-up box with some help text, but the original help topic text remains behind the pop-up help box.

- Four navigation facilities are provided to the end-user:

    1. Hyperlink traversal: Some phrases or words in the help text are marked as

hotspots. Clicking on those hotspots will display new help topics related to that hotspot.

2. Indexed traversal: All hotspots are arranged in an alphabetically sorted order. The user can go to any of these by clicking on an item in the index list. This is similar to the index at the end of any printed manual.

3. Lateral movement in the document: This navigation is similar to the one used to read printed documents. Topics or sections are organized in some predefined order. The user can go to the next topic and previous topic by pressing the NEXT and PREVious button provided in the help screen.

4. History traversal: The help system builds a list of topics the user has visited since the invocation of the help system. The user can now go up or down this list using the BACK or FORWARD button provided in the help screen.

- Window resizing: When the user resizes the screen, all the help text that is currently shown is re-formatted to fit new screen dimension.

- Scrolling: When there are more text than could fit on the screen the user can use the scrollbar to see the hidden text.

- Display of pictures: The help system can display pictures or icons.

- Multiple fonts and color support: The help text can have multiple font-types and colors.

- Print facility: Allows the user to print the help text currently shown on the screen to the printer.

5.2.3 Help System Design

The on-line hypertext help system has two modules, a help compiler and a run-time module. The help compiler is used to develop help documents by the on-line help developer. The run-time module is utilized by the end-user to review the document when needed. The next paragraphs describe the help compiler followed by a description of the run-time module. Figure 5.3 depicts the architecture of an on-line hypertext help system.

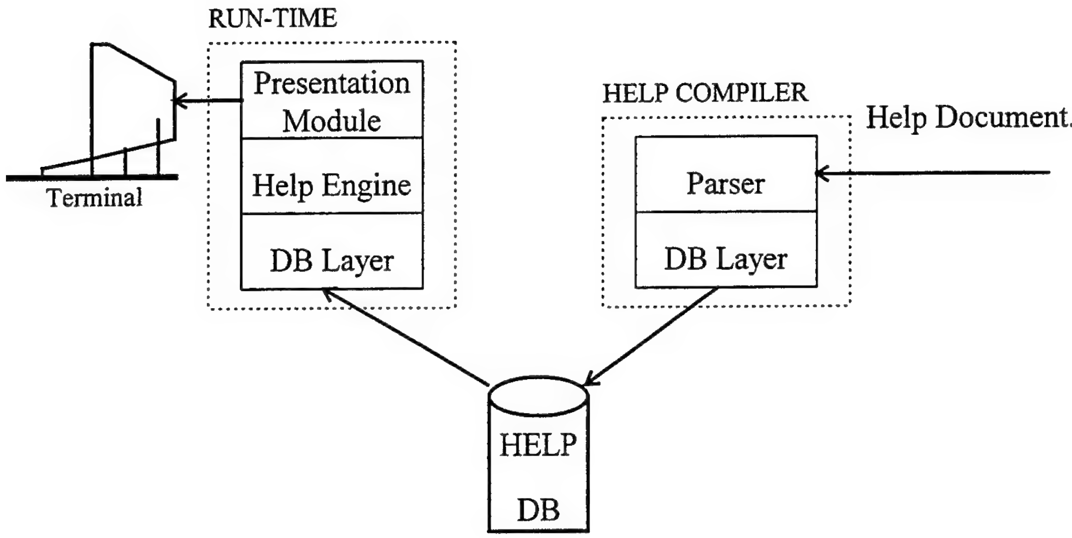

Figure 5.3 On-line HyperHelp systems architecture.

5.2.3.1 Help compiler

The help compiler is an utility which reads the help input file entered by the help authors, parses the help text, and populates the help database. This help database will be accessed by the run-time help module to display the help text when the user request for it. First of all, the application developer or whoever is responsible for setting up the help system, should create a help document. There is much literature on designing and writing on-line documentation [27] [28]. Assuming the help text is in an ASCII file, the next step in the help-authoring process is to insert help keywords in the help text file. These keywords identify help topics and hypertext, establish links between topics, and define paragraph layouts including font size, width, and color of text etc. in the help document. These help keywords are derived from a popular word processor in the UNIX environment, Frame-maker. The keywords are part of Frame-maker Markup Language (FML) used in FrameMaker. One of the main features of FML is that it provides a few predefined keywords and lets the user define new keywords, called user-defined keywords. Once the user defines a keyword, from that point onwards the keyword can be used as any other system-provided keyword. All FML keywords are enclosed within angled brackets. In help system a subset of the FML keywords is used. Following is a listing of keywords recognized by the help compiler.

<*!DefineFont* font_keyword

<*Family* font_family_name>

<*pts* font_point_count>

<*Bold* | *Italic* | *Regular*>

>

<*!DefineColor* color_keyword

<*Cname* color_name>

>

<*!DefinePara* para_keyword

<*FirstIndent* point_count>

<*LeftIndent* point_count>

<*RightIndent* point_count>

>

<*Hyper* { *replacement* | *popup* } hyper_index hyper_text>

<*Picture* picture_type position_from_left_window_border picture_width picture_height picture_file_name>

<*Comment* Help authors comment goes here>

Note:- Italic words are FML keywords and the rest are user definable.

Family - defines the font family, for example, courier, times etc.

pts - refers to the point size of the font, for example, 10pts, 15pts.

Bold | *Italic* | *Regular* - refers to the font style.

!DefineColor - used to define colors

Cname - one could use any named colors, for example, red, blue, yellow etc.

!DefinePara - defines a paragraph format.

FirstIndent defines indentation of the first line in a paragraph.

LeftIndent marks the left margin of a paragraph.

RightIndent identifies the right margin of a paragraph.

Hyper identifies hyper text. Hyper text are highlighted text in the screen, when clicked using mouse device it will refresh the screen with new help topic.

replacement, replaces the current help topic with the new help topic. And the new help topic is identified by hyper_index. It is an option which follows *Hyper* keyword.

popup, display text associated with hyper_index in a popup window with the current help topic still displayed in the background of this popup window. This is similar to *replacement*, it follows *Hyper* keyword. But only one of *replacement* or *popup* could be mentioned for a hypertext.

Picture, is used to insert picture within an help document. It has 5 mandatory arguments, they are picture type, for example XPM, GIF, TIFF picture formats(currently it help system support only XPM picture format, so any picture type other than XPM will prompt error during help compilation); left indent of the picture from the window border; width of the picture; height of the picture and the name of the data file containing the picture.

Comment, is provided as a means for help authors to insert their comment for later reference.

In the para-definition segment, the user can have any user-defined keywords and FirstIndent, LeftIndent, and RightIndent. The system assumes a default value if any of these is omitted in paragraph definition. In the font-definition section, the help author could have Family, pts, and either one of bold, italic, or regular keyword. The subsequent paragraph discusses the DB layer and database schema design for hypertext help system after the steps involved in parsing FML files are enumerated.

Compilation of the help document is a two-step process:

- **Parsing the ASCII Help Document File Containing FML Keywords**: This step checks for syntax errors and stores the help document in in-memory data structures. Typically, a help document will have all paragraph layout, font, and color definition features in the beginning of the document. These are referred to as formatting information. The run-time module will use this formatting information later to nicely format help texts when invoked by the end-user. The help compiler is a single-pass compiler, so keywords should be defined before being used in the help document. The help compiler parses as it reads a line at a time from the disk ASCII file. Data structures in the memory are populated as paragraphs, after paragraphs are syntactically checked. Parsing of the help document is successfully completed when the help compiler reaches the end of the file without any syntax errors.

- **Populate database**: In this step, the help compiler extracts data from data structures populated in the previous step and loads it in the database, using the

database provided API (Application Program Interface) calls. The hypertext help system consist of seven tables. The design and relationship of those tables is discussed in the next paragraph.

5.2.3.2 DB Layer & Schema Design

One of the main factors in designing the DB layer is to be able to port the help system across various commercially available databases [26]. This is done by not using the proprietary features of any database, thereby the source code is highly inter-operable. Currently the help systems database layer works with Informix, Sybase and DESc. The database layer is written in C using respective DB API calls provided by each of the databases mentioned above. The next paragraph discusses the schema design of the help system.

There are seven entities in the help system:

- Application
- Para_Header
- Para_Line
- Para_Display
- Color
- Font
- Index Table

The Application entity is at the top of the hierarchy. There is one record or row for every application's help document. The Para_Header stores help topics of an application's help document. The help topic is the basic building block in a help system. A help topic consists of one or more paragraphs with different formatting layouts. Also, there is at least one or more help topic associated with every application, so there exists a one-to-many relationship between the Application and Para_Header. Some database management systems do not allow the character data field to exceed 256 characters, but a help topic could be more than 256 characters. To handle this, another entity was created called Para_Line, and a 1-to-M relationship was established between Para_Header and Para_Line. A paragraph in a help document exceeding the 256 character limit is split and stored in the Para_Line entity. The number of records in Para_Line for a single paragraph in a help document is $\lceil$total # of chars in a para / 256$\rceil$. All paragraph formatting information goes into the Para_Display entity. Records in Para_Display are referred to Para_Header for formatting details. The color and Font entities have color and font definitions used in the help document, referred to in Para_Display. Figure 5.4 shows all the entities content of all the entities and the relationship between them.

5.2.3.3 Run-Time Module

The run-time module is used by the end-user when invoking on-line help from a GUI application. The run-time module retrieves the help text from the database populated by the help compiler, formats according to the formatting information stored in the help database and displays the text to the end-user. There are three layers in the run-time module of the hypertext help system:

- Presentation Layer
- HyperHelp Engine
- Database Layer

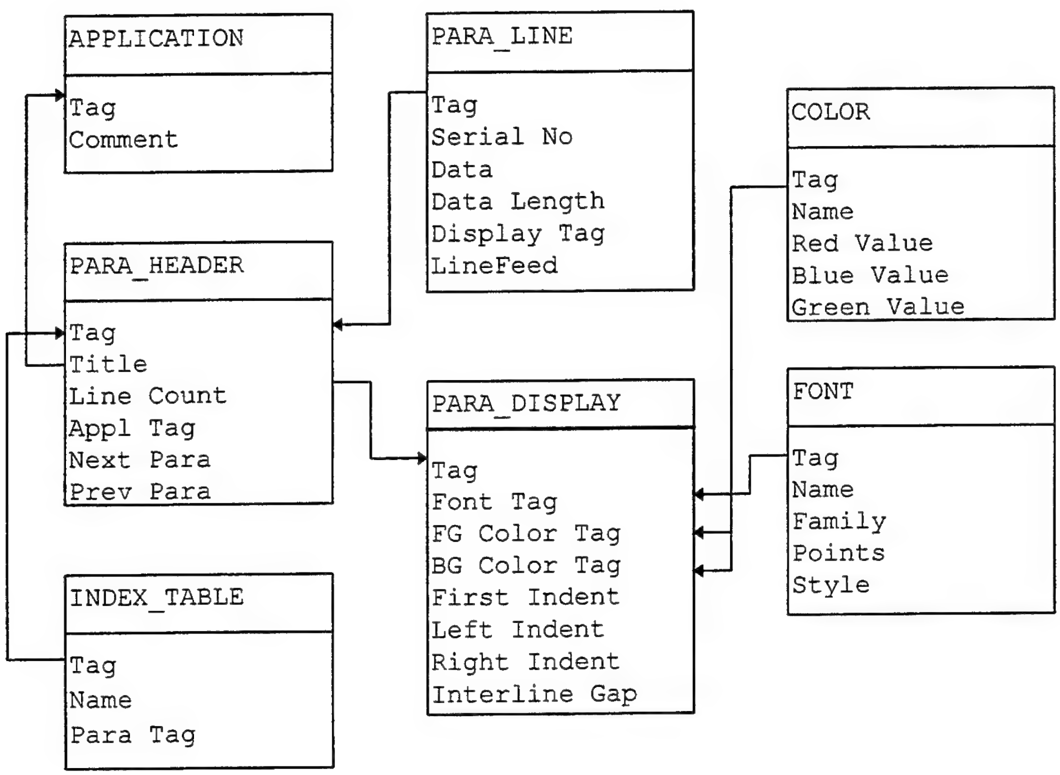

Figure 5.4 E-R diagram for help system.

The database level concerns traditional issues of information retrieval. The HyperHelp engine is responsible for formatting text and keeping track of user actions. It is the bridge between the database and presentation layers. The presentation layer interacts with the end-user; it is responsible for creation and maintenance of screens necessary for displaying the on-line help document. This layer also reformats the help text when the window is resized; handles scrolling, waits for user action, and passes user input to the help engine. The HelpEngine is the layer which determines the cause for user action; if necessary, it retrieves more data from the database or supplies data from its own data structure to the presentation

layer for display.

5.3 UIL2C Design

The UIL2C is a utility developed for DAKOTA. It is used to recognize the user interface language (UIL), check for syntax and semantic errors, and generate a C code that would be later compiled and linked with other X source files to create X client object file. UIL is a screen layout specification language, which enables one to quickly create and easily maintain screens in the X window environment. The UIL script can be either created by the user using any editor or can be generated from by a User Interface Management System (UIMS), such as DEC-VUIT. Apart from all its advantages there are some disadvantages which prompted development of UIL2C. When X client is run, it consults with the screen specification file to get display characteristics. The process can be slow when there are more widgets/screen objects needing to be loaded. Also, dynamic adding and removing of user events to any widget is impossible. DAKOTA requires dynamically adding and removing user events at run-time. To circumvent these problems, a UIL2C compiler was developed to convert UIL code to its equivalent C source code. In the next few paragraph we will discuss about UIL2C design.

There are two main parts to every display object that is defined in a UIL file; they are arguments and controls. The arguments part of an object describe the object's display attributes, such as width, height, x position, y position, etc. Each object type permits a set of attributes. For example, XmNheight is an attribute to most objects and has an integer data type. To specify height for a widget, the attribute name XmNheight can be used and an integer value is specified in the argument part of the object. The controls part of an object defines which objects are children of, or controlled by, a particular object. This part basically defines the object hierarchy in a screen. A sample UIL script is given below:

```
object_name : widget_class
  {
  arguments
    {
      attribute_name1 = attribute_value1
      attribute_name2 = attribute_value2
              . . .
      attribute_namen = attribute_valuen
    } ;
  controls
    {
      widget_class object_name1
```

```
        widget_class object_name2
                ...
        widget_class object_namen
    };
}
```

where,

attribute_name[1,2...n] are one of those pre-defined display attribute values

attribute_values[1,2...n] corresponds to their respective attribute_name

object_name[1,2...n] are one of those user defined object names

The UIL2C, converts UIL file to C code in a three-step process:

- First, the input UIL file is read and parsed for syntax errors. This part is done using the UNIX lexical analyzer and yacc LR(1) parser. During this process, a singly linked list of objects is created, with each node in the list pointing to an attribute list and to a list of child objects. This data structure is later referred to in the semantic checking and code generation stages.

- The second step in the conversion process can be divided into two stages of semantic checking, that are needed to make sure the of validity of the UIL files content. First, validate if the object widget class has a valid attribute or resource name in the argument part. For example, XmNlabelString is an attribute name; it applies to only label widget class. If widget class is anything other than XmLabelGadget or XmLabelWidget, having XmNlabelString in the argument section for an object is a semantic error. Then, check if all the widget class defined in the control part is appropriate for this object. For example, a single line text field widget cannot have form or bulletinboard widget, as its children. Information about the list of all possible attributes for all widgets, and the list of all possible children for a particular widget, is stored in two separate ASCII data files. The UIL2C refer to these data files while verifying for semantic errors. File formats and a sample of these data files are given in Appendix D.

- Finally, UIL2C generate C source code, if both the syntax and semantic checks completes successfully. A sample UIL file and its equivalent C source code is given in appendix-D.

5.4 Integration of Sub-Systems

DAKOTA is comprised of many sub-systems. To be able to perform parallel development of different parts of the system, a modular design approach [24] as in software engineering technology was used. A modular design reduces complexity, facilitates change, and results in easier implementation by parallel development of different parts of a system. The concept of functional independence evolved from modularity and the concept of abstraction and information hiding [25]. Software system was designed so that each module addressed a specific sub-function of requirements and has a simple procedural interface when viewed from other parts of the program structure. There are four major software

components developed as part of the scheduling system: manual scheduling, automatic scheduler, hypertext help system and database system. DAKOTA invokes functionalities in all these modules in solving scheduling problem. Each of these components is made up of sub-components. Since this paper concentrates on the manual scheduling and help system we will list the sub-components in these modules and describe the order of module integration undertook to reach the final software environment. Adapted bottom-up integration strategy as in software engineering terminology. The various sub-components or components are first numbered and categorized based on the input, output, and processing nature of the module as below:

1. Presentation layer	Input, Output
2. Presentation logic	Processing
3. Data logic	Processing
4. Database & ASCII File interface	Input, Output
5. Help system DB layer	Input, Output
6. Help compiler	Processing
7. Help system presentation layer	Output
8. Database (DESc)	Processing
9. Optimizer	Processing

First, the input and output modules are developed and unit tested. To test the input and output modules drivers and stubs were written which simulated the back-end processing logic. Then, developed the processing module. Figure 5.5 depicts the order of development. Each numbered box corresponds to the above list. Lines from two boxes meeting at a node represents the integration order.

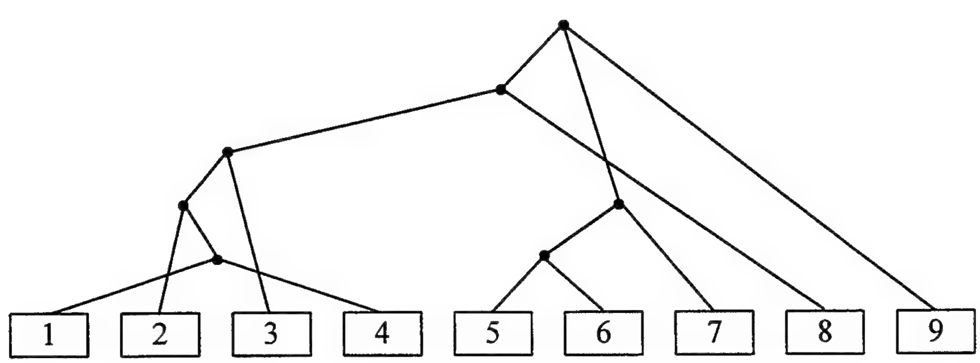

Figure 5.5 Bottom-Up Integration of the sub-components.

To increase modularity and maintainability, the user interface and the application code should be separated as much as possible. This is not easy, especially in graphical applications with direct feedback, where even low-level functionality (such as dragging) requires some processing by the application side. The application side is made up of interconnected conceptual processes supported by data stores/structures, and the interface side is represented by a hierarchy of interface objects (called widgets). A dialog layer in between keeps both of the layers consistent. The dialog layer is most of the cases is thin because the application structure is modeled directly to aid the user interface. There is also a thin coupling layer which the user interface uses to manipulate data in the database. The database interface layer was designed in such a way that DESc can be replaced with any other commercially available databases, such as ORACLE, Sybase, etc.. This DB interface layer uses DESc API function calls to query or manipulate data. Output from this layer is a data structure that contains necessary data to either display to the user or to be used for some computation.

Finally, the optimizer is integrated with the user interface loosely through two procedure calls. First, SP solver is called to heuristically determine columns representing the best assignment of request legs to missions. Then, insertion algorithm is called to recreate schedules for selected assignments [5]. The data logic layer in DAKOTA handles transferring the problem definition data and the schedule output between Optimizer and the user interface. There is on-line help button available for every DAKOTA screen. When the user presses the HELP button on the window or dialog box, the button's callback procedure invoke a help system entry function. Depending on from where this procedure is invoked, a corresponding help topic is passed as a parameter to the help system, which loads that particular help topic in a window. Scheduling application communicates with help system using one single procedure call: XHTHelp("<topic_name>"). This little coupling and functional independence between the sub-systems makes DAKOTA an easily maintainable and scaleable software system.

6. EFFECTIVENESS OF THE GUI

One of the greatest promises of a functionality-rich scheduling system is that it offers new methods of solving problems. Without adequate system support, the chances of a user finding an optimum solution for a given problem are small. With this in mind, DAKOTA was built with many functionalities to help users find an optimum solution with ease. This section discusses the valuable features in MAP, SCHEDULE and CANDIDATE SOLUTION viewing screens and how those features help in solving scheduling problems.

The map screen displays a map of the whole world, including country borders, rivers, lakes, bases, base labels, etc.. At any time the screen shows a day fleet activity. The user can select a date by typing the date in a text field at the top left corner of the map screen. It will retrieve the missions and unscheduled requests falling in that date from the database. This information is displayed in the scrollable box on the right-hand side of the map screen. The same information is also shown on the map, using lines to connect bases in mission and unscheduled request legs. Missions are shown on the map as solid lines and unscheduled requests as dashed lines. If there are lot of lines on the map, it could clutter the map and obstruct or deviate the user's attention from the data of interest. To circumvent this problem, the user could zoom in on an area and view only some missions or requests of interest. There are multiple levels of zooming available. One can zoom-in on an already zoomed-in map. It is also possible to zoom out to the previous zoom level or to the original map view. This feature lets the user totally control which part of the data shown at anytime.

Another valuable feature available in the map screen is the ability to see only part of a day's fleet activity, thereby eliminating unwanted data while analyzing the data of interest.

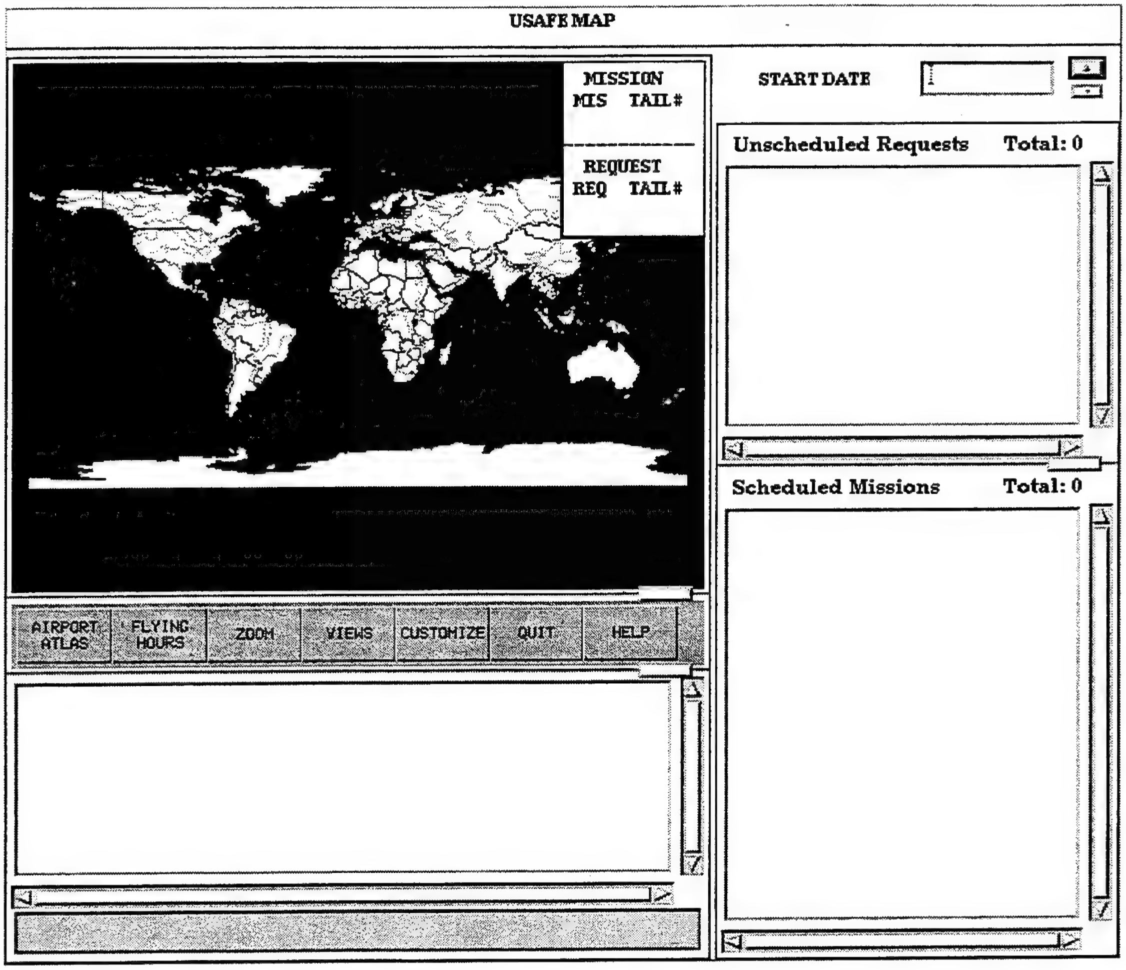

Figure 6.1 Map screen.

There are three ways this function can be achieved:

- View a particular type of aircraft activity: Selecting VIEWS->BY_PLANE_TYPE, pop up a dialog box with the list of aircraft in the fleet. Select aircraft that are of interest by highlighting one or more aircraft. Once finished with the selection, accept by pressing the OK button. This will redraw map with missions flown by selected aircraft.

- View only selected missions: Selecting VIEWS->BY_SPAR#, pop up a dialog box with the list of missions flown on a particular day. Highlight or de-highlight by clicking on the mission list. When finished with the selection, press the OK button to redraw the map with the highlighted missions information.

- View either missions or unscheduled requests: View only missions or only unscheduled requests by checking the radio-button in the CUSTOMIZE dialog box (shown in Figure 3.1). This can be reached by selecting CUSTOMIZE->MAP_DISPLAY in the map screen. The customize dialog box also allows turning ON or OFF landmarks like country borders, rivers, lakes, etc..

Another feature in the map screen is to be able to perform various queries on airbases. There are six different queries that can be performed on the atlas database. Pressing AIRPORT_ATLAS in the map screen will pop up a dialog box, where one can input ORIGIN base, DESTINATION base, AIRCRAFT TYPE, and RADIUS in miles. Some of these entries

need to be filled depending on a query type. After typing the required information, press the OK button to execute the query, which will retrieve data from the database and update the map screen accordingly. Following are various queries:

- Distance query: Finds the distance between any two given bases in air-miles. It displays the selected bases on the map.
- Time query: Given two bases and an aircraft type, it finds the time taken to fly between bases.
- Radius query: Displays all the bases that fall within a given radius of a base.
- Range query: Displays all the bases within the reach of a given aircraft from a base.
- Fuel stop query: Gives a list of bases that are in range of both the origin and destination for a given aircraft type. This list is ordered in terms of the most efficient fuel stops first.
- Fuel path query: Displays the shortest route to travel from origin to destination for a given aircraft type through the closest fuel stops on the flight path. The shortest path algorithm used in finding the fuel stops provides an approximate estimate. With the huge number of possible fuel stops within the vicinity of the flight path, an exact algorithm would take a longer time to find the best results.

These features come in handy while scheduling in the schedule screen. The user is enabled to find a fuel stop located along the path between origin and destination, find a stop within the aircraft's endurance from a base, find all bases within the given radius of a base, etc..

6.1 Candidate Solution Screen - Cellular Graph

The candidate solution viewing screen displays the schedule in three different formats. Data necessary for display are obtained from a disk data file. This file is created in the scheduling screen by selecting the save option from the file pulldown menu. All the missions and requests in a scheduling session are saved in the disk file. The candidate solution screen lets the user compare two or more solutions and choose an optimum one to permanently save to database. There are three parts to the candidate solution viewing screen, with each portion corresponding to one format of the schedule display. The left top portion of the candidate solution screen shows the schedule in a map format similar to that of the map screen, with bases connected and color-coded for every mission/request. The right top part shows all the missions and unscheduled requests in a tabular text format. The bottom portion displays a cellular graph of the missions, with time on the X axis and mission identifier on the Y axis. The cellular graph shows the duration of an aircraft in-flight and ground time from the start to end of a mission. A sample screen is given in Figure 6.2.

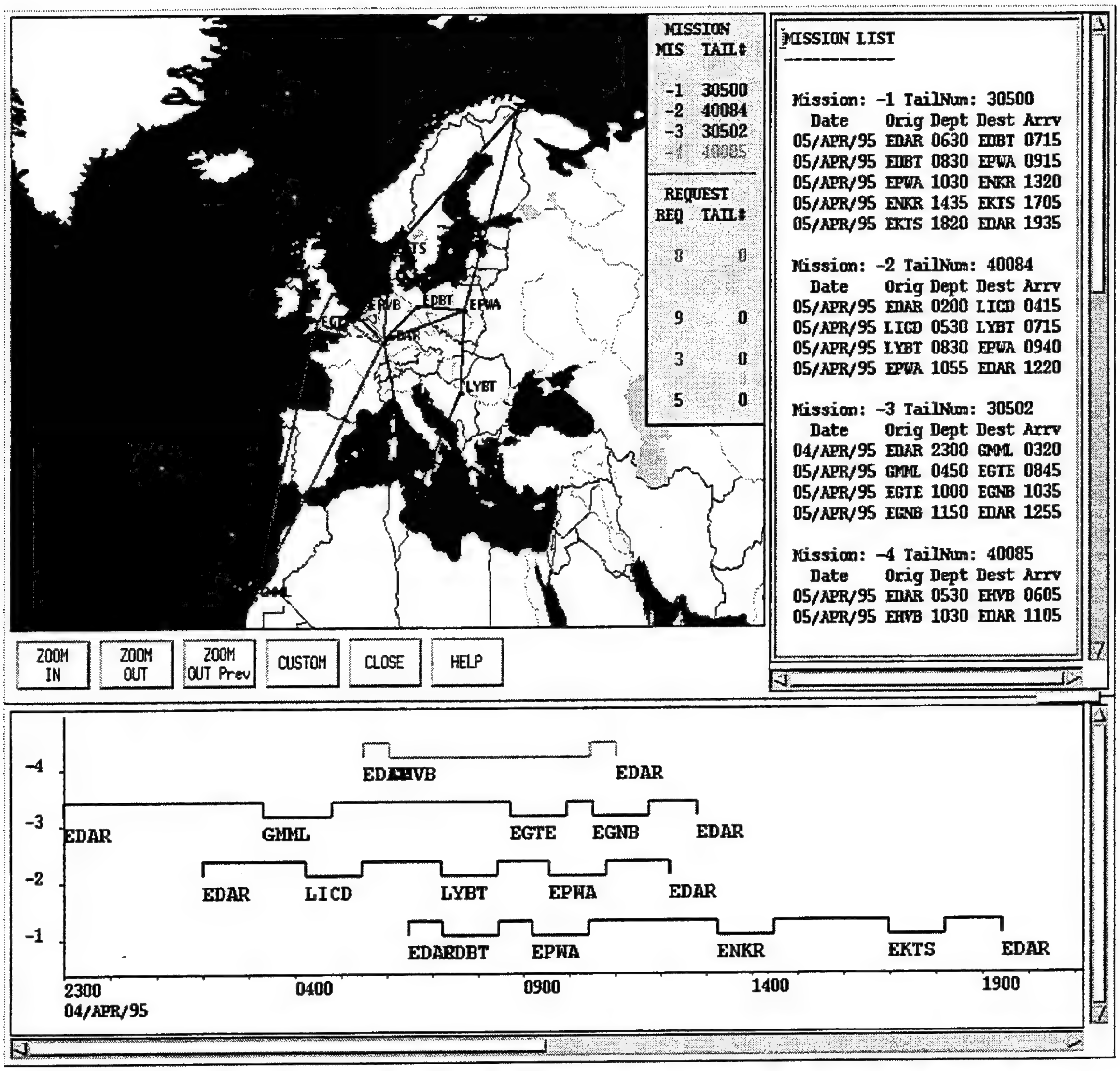

Figure 6.2 Candidate solution screen.

There are a few customized options provided for changing the display attributes of a cellular graph in the candidate solution viewing screen. These options are given in command line as follows:

CandidateSolution -input <file_name> [-x_scale <#_pixels> -y_scale <#_pixels> -start_x <#_pixels> -start_y <#_pixels> -uptime <#_pixels>]

where,

input - name of the disk file containing the missions and request of a scheduling session

x_scale - number of pixels per hour

y_scale - number of pixels between missions

start_x - number of pixels from the left screen border

start_y - number of pixels from the bottom screen border

uptime - number of pixels between ground and in-flight of a mission

7. Conclusions

DAKOTA was developed in response to a specific scheduling need for USAFE executive airlift. First, the problem domain was carefully analyzed as well as all the factors affecting the scheduling paradigm. From this analysis, the software system architecture was designed. Then, a mockup or prototype of the GUI system was built and demonstrated to the end-user at USAFE before going into full-fledged development of the scheduling system. Initial prototyping of the system helped identify detailed input, processing, and output requirements. Also, a primitive DOS graphics-based scheduling system used at USAFE assisted the developers to learn from mistakes of another software product and fine tune the scheduling environment to meet end-user needs.

This project detailed various GUI design principles. The GUI concept that best suited for the scheduling environment were implemented. Also, the user interface style was adapted dictated by Open Software Foundation (OSF) for Motif applications. Therefore, anyone who is familiar with GUI conventions will quickly become accustomed to our software system without having to go through any training. There are many features built into the system to enable efficient scheduling. These features are modeled to aid specific tasks in the problem domain. While scheduling the user does not have to make a note of any information on paper, use a calculator to perform distance or time calculations, or browse through any printed manuals or books for information on procedures and rules of operation. All the tools necessary for scheduling are bundled with DAKOTA. For example, the user utilizes scheduling screen to schedule a request, in the background the user can have the map screen opened and initiate queries for distance between any two ICAO's, determine the time taken to travel between any two ICAO's, locate bases within a given radius of an airport, etc. A user manual is stored in the help database and can be accessed using the on-line help system; therefore, the user does not have to reach out for any printed document. All tools and techniques required during a scheduling process are at the fingertips of the end-user. A major achievement was made in integrating the manual and automatic scheduler. The confidence is boosted among users by being able to schedule many requests within a short time, using the decision support model, and still being able to edit and fine tune manually.

Even though DAKOTA is rich in functionality, there may be new features that need to be added in the future. Since this system was developed using a third-generation language, the size of the software system is huge, approximately 50,000 lines of code for the GUI alone. This does not include the DESc database, Optimizer and on-line help system. Rehosting, using a fourth-generation language, third-party tools or environment could reduce the size of the source code and ease software maintainability.

DAKOTA generates three reports. The end-user needed to be able to dynamically define and produce new reports from the system. This capability is needed, and is natural extension of the software system. An approach for future enhancement would be providing the capability of writing files that can be read by other software systems commonly used for reports and presentations, such as spreadsheets (e.g., Excel, Lotus 1-2-3, and QuatroPro) and presentation graphics systems (e.g., PowerPoint, Persuasion).

References

[1] Mountford S.J. et al. "Drama and Personality in Interface Design" Proceeding of CHI 1989 conference on "Human Factors in Computing Systems". ACM Press.

[2] J.G. Neal and S.C. Shapiro. "Architectures for Intelligent Interfaces Elements and Prototypes" Proceeding of CHI 1988 conference on "Human Factors in Computing Systems". ACM Press.

[3] Jerry Manheimer, Rodney Burnett and Jo Ann Wallers. "A Case Study of User Interface Management System Development and Application" in CHI 1989. ACM Press.

[4] Whiteside, J., Bennett, J., and Holtzblatt, K. Usability engineering (1988). "Our experience and evolution", Handbook of human computer interaction. ACM Press.

[5] Richard S. Walker(1995). "SCHEDGEN: Heuristics for Customized Airlift Resource Allocation and Scheduling". Department of Computer Science & Operations Research, North Dakota State University, Fargo, ND.

[6] Prabhukumar, Ganapathy (1993). "DESc: A Database Engine for Scheduling Applications". Department of Computer Science & Operations Research, North Dakota State University, Fargo, ND.

[7] Motif 1.2 Style Guide (1993). SunSoft, Sun MicroSystems, Inc.

[8] The Windows Interface: An Application Design Guide (1992). Microsoft Press, Microsoft Corporation.

[9] Xlib,Xt and Motif Programming Manual (Vol 1, 4, 6) (1991). Oriely Publishing Co..

[10] Arnaud Le Hors & Colas Nahaboo (Nov 1991). "XPM: The X PixMap Format" User Manual. BULL Research, Paris, France.

[11] Marcus, Aaron (1990). "Corporate Design Principles for Graphical Human Computer Interfaces,". Prentice Hall.

[12] Kieras, D.E. and Polson, P.G (1985). "An approach to the formal analysis of user complexity" in International Jornal of Man-Machine Studies.

[13] Marcus, Aaron (1982). "The Computer Image: Applications of Computer Graphics". Addison-Wesley Publishing Co., MA. pp76-90.

[14] Marcus, Aaron (1985). "Users Must Establish Own Rules for Colors". Computer Graphics Today, 2(9). pp 7ff.

[15] Donald A. Norman & Stephen W. Draper (1986). User Centered System Design. Lawrence Erlbaum Associates, Publishers.

[16] Harold Thimbleby (1990). User Interface Design from ACM Press.

[17] Carroll J.M. (1984). Minimalist design for active users. pp 39-44. Proceeding Human-Computer Interaction-INTERACT'84 (Shackel B. ed.)

[18] Pike R. (1988). Window systems should be transparent. Computing Systems, 1(3), 279-96.

[19] Hanan Samet (1989). "Applications of Spatial Data Structures". Addison-Wesley Publishing Co. pp 225-260.

[20] Greg Kearsley (1988). "Online Help Systems - Design & Implementation". Ablex Publishing Corp. pp 73-83.

[21] William Horton (1990). "Designing & Writing Online Documentation Help files to Hypertext". John Wiley & Sons.

[22] Norman, Kent L., Linda J. Weldon, and Ben Shneiderman (1986). "Cognitive layout of Windows and Multiple Screens for User Interfaces". International Journal of Man-Machine Studies.

[23] Knuth, D. (1973). The Art of Computer Programming (2nd ed.) Fundamental Algorithms (vol. 1), Sorting and Searching (vol. 2). Addison Wesley.

[24] Roger S. Pressman (1992). "Software Engineering - A practitioner's Approach" (3rd edition) from McGraw-Hill, Inc.

[25] Bruce I. Blum (1992). "Software Engineering - A Holistic View". Oxford University Press.

[26] BrockMann, R. John & William Horton (1989). "From Database to Hypertext: An Introduction Odyssey". Cambridge MA: The MIT Press.

[27] Bradford, Annette Norris (1984). "Conceptual Differences between the Display screen and the Printed Page". Technical Communication.

[28] Bradford, David (1984). "The persona in Microcomputer Documentation". IEEE Transactions on Professional Communication.

APPENDIX A-MASTER DATA ENTRY SCREENS

DAKOTA handles a large amount of lookup data which is used during a typical scheduling session. There are three master or lookup entity: airport atlas, passenger and aircraft details. We have developed GUI screens to handle these master tables, with one screen for each lookup record management. In this Appendix, the attributes are listed in each of those master record with a sample screen.

Airport Atlas Data Entry Screen: The airport atlas screen is used to maintain atlas data. An atlas record consists of information given in Table A.1.

Table A.1 Airport Atlas Attributes

Attribute	Description	Valid data
ICAO	A unique code for an airport all-over the world.	Character of length 4. It is a mandatory attribute.
IATA	Code	Character of length 4.
CITY NAME	City where the airport is located	Character of length 30.
AIRPORT NAM	Name of the airport	Character of length 30.
LATITUDE	The Latitude of the airport	Real value between -90 and +90.
LONGITUDE	The Longitude of the airport	Real value between -180 and +180.
LRUNWAY	Length of the runway	Integer between 0 and 100000.
ELEVATION	Angle of elevation	Integer between 0 and 100000.
STD DATE	Date to convert to standard time	Date as DD/MMM/YY.
SAVE DATE	Date to convert daylight	Date as DD/MMM/YY.

	savings time	
STD HOURS	Number of hours to move from Standard (Zulu) time	Real between -12.0 and +12.0.
SAVE HOURS	Number of hours to move from Daylight savings time	Real between -12.0 and +12.0.

Using the atlas data entry screen the user can add new atlas records, delete an existing atlas records, and make changes to any of the attributes (see Table A.1) in an existing record. The changes made in the screen are first checked for validity of the data before permanently saving to database. A sample atlas screen is given in Figure A.1.

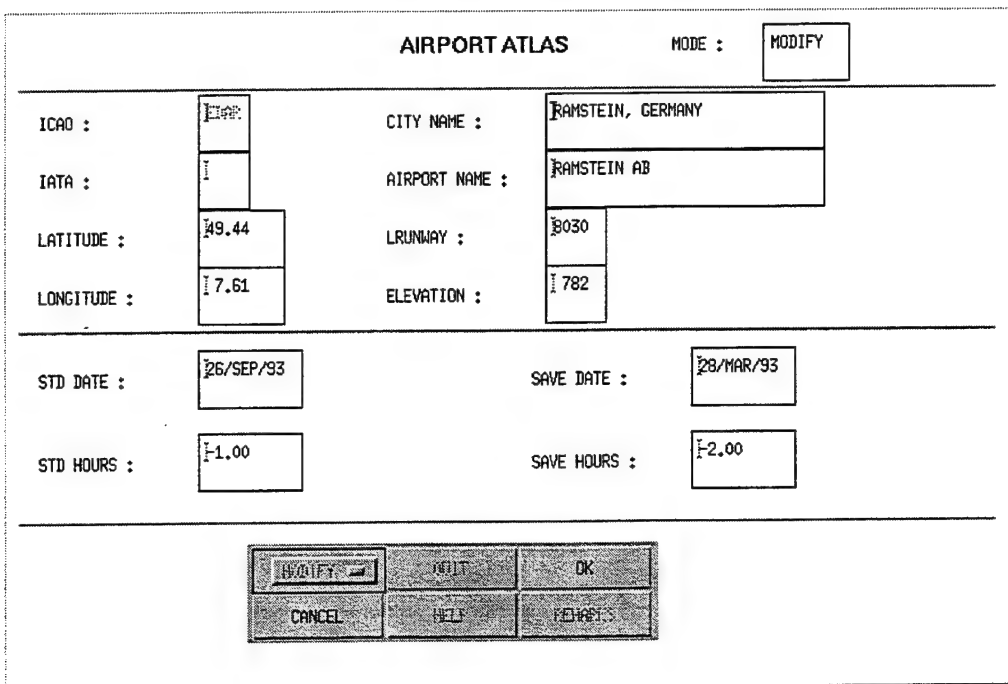

Figure A.1 Atlas Data Entry screen.

<u>**Aircraft Data Entry Screen:**</u> The aircraft screen is used to maintain aircraft data. An aircraft record consists of information given in Table A.2. Using the aircraft screen the user will be able to add a new aircraft type, delete an existing aircraft type and make changes to any of the aircraft type attributes (see Table A.2). Changes made on the screen are first checked for validity of the data before permanently saving to database. A sample aircraft data entry screen is given in Figure A.2.

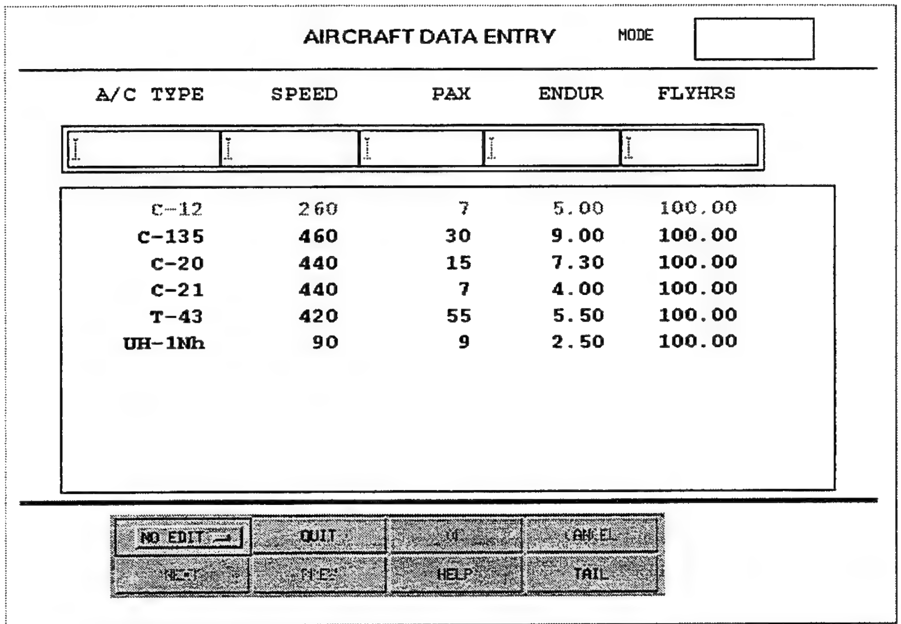

Figure A.2 Aircraft data entry screen.

Table A.2 Aircraft Attributes Table

Attributes	Description	Valid data
A/C TYPE	Name of an aircraft type.	Character of length 10. A mandatory field.
SPEED	Speed of the aircraft.	A positive integer.
PAX	Capacity of the aircraft in terms of number of passengers.	A positive integer.
END	Endurance of the Aircraft. Continuous time in flight in terms of hours.	A positive integer.

FLYHRS	Budgeted flying hours per fiscal year.	A positive integer.

At the bottom of the screen, in Figure A.2, the two rows of buttons facilitate various operations on the aircraft data entry screen. At any time during the data entry operation, pressing the OK button confirms the changes to the database, while the CANCEL button abandons the changes.

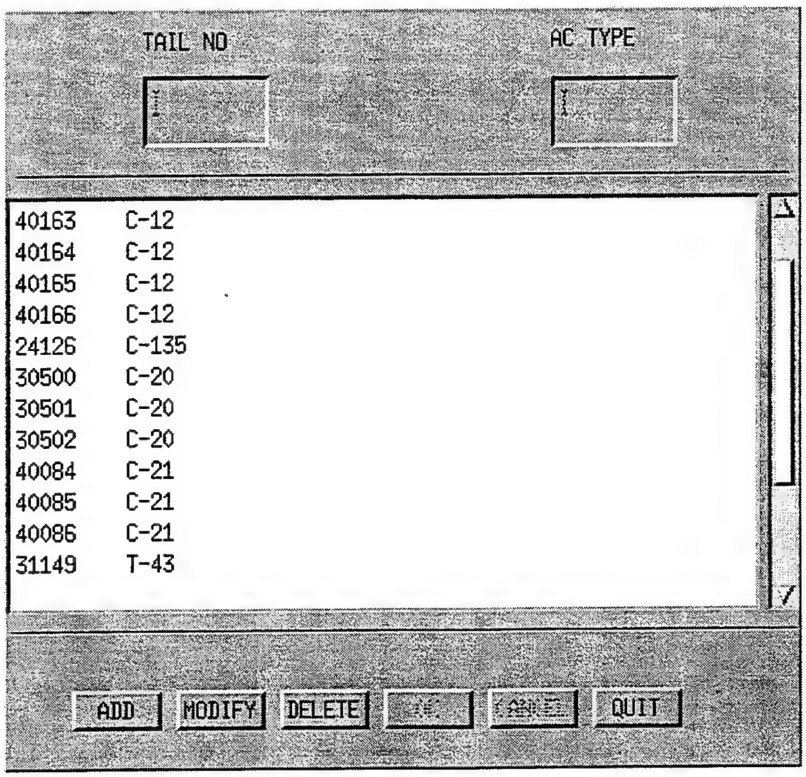

Figure A.3 Tail number management screen.

There can be more than one physical aircraft to an aircraft type. Each physical aircraft is identified by a number called the tail number. The user can add or remove a physical aircraft from an aircraft type using the screen given in Figure A.3. To add a tail number to an aircraft type, select the ADD button from the row of buttons at the bottom of the screen in Figure A.3. Then type in the tail number and aircraft type in the editable text field at the top half of the screen. To confirm the change, press the OK button. To modify or delete a tail number from an aircraft type, first select a row from the list box and press either the MODIFY or DELETE button. In modify mode, the user can change a tail number or associate a tail number to a different aircraft type, or both. Once done with the change, confirm it by pressing the OK button or CANCEL to discard. In delete mode, pressing the OK button will

remove that tail number from the database permanently.

Passenger Data Entry Screen: The passenger screen is used to maintain passenger information about who has flown in the past or who is likely to fly in the future. A passenger record consists of information given in Table A.3. A sample screen is given in Figure A.4. Using the passenger screen the user can add, modify, or delete passenger record from the database. There are two parts to the passenger data entry screen. One on the left-hand side contains the list of all passengers in the database. Editable text fields on the right-hand side are used to add or modify attributes related to a passenger record. To add a new passenger to the database, first select the ADD button; then, enter all passenger attributes and save to the database by pressing the OK button. To modify or delete a passenger record, select the MODIFY or DELETE button, respectively. Then click a passenger name in the list box on the left-hand side; this will retrieve all attributes (see Table A.3) and populate in the editable text field on the right-hand side. In modify mode, make changes and select the OK button to save to the database or select the CANCEL button to discard the any change. In delete mode, select the OK button to delete the retrieved passenger record from the database permanently.

Table A.3 Aircraft Attributes

Attributes	Description	Valid data
L NAME	Last Name of a pax	Character of length 30.
F NAME	First Name and middle Initial	Character of length 20.
TITLE	Job title of the pax	Character of length 10.
ID	Identification. A passport #, or SSN#	Character of length 11.
DV CODE	Rank of the officer	Character of length 2.
PHONE	Contact Phone number	Character of length 20.

SERVICE CAT	Service Category: (Army, Navy, Air Force, Civilian)	Character of length 1.

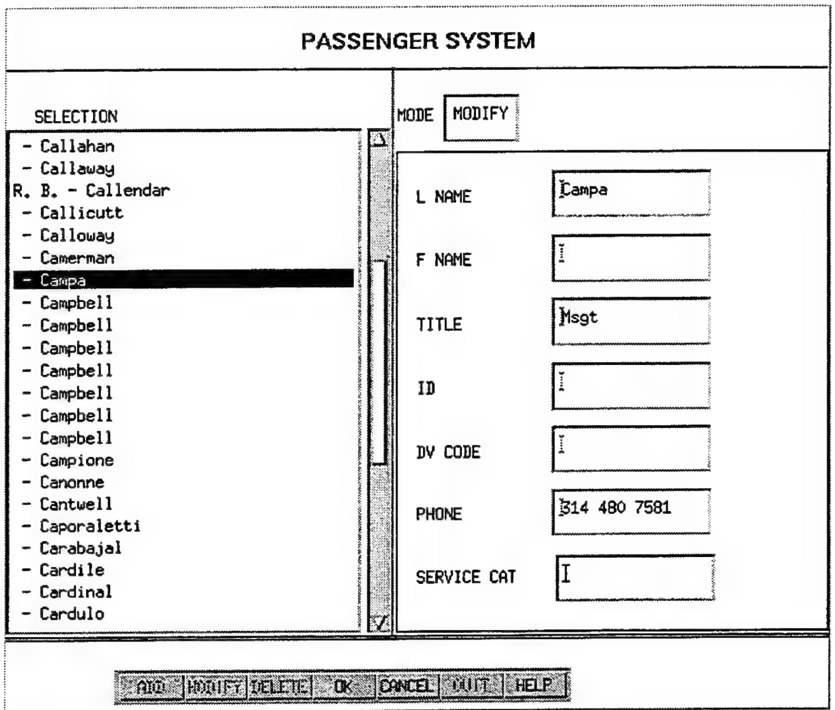

Figure A.4 Passenger data entry screen.

APPENDIX B-SCHEDULING FILE FORMAT

Since a scheduling session could be prolonged, the scheduling session needed to be able to be saved in a disk data file temporarily in order to continue the process at some later point in time. In this section the data file format is described and how to save and load a scheduling session. The in-memory version of a scheduling session is saved in four different disk files. Two files have the data as retrieved from the database, and other two files have data with human and automatic scheduler changes. Of these two files in each category, one is an ASCII file and the other is a binary file. The ASCII file contains the file offsets in the binary data file for all requests and missions; it is also called the control file. The binary file contains actual request and missions data. The save program will compute the file offsets based on the size of the request and mission data structures, and first, create the control file. Then, the in-memory data structure is written to the binary file in the previously computed file offsets. The load program will just do the opposite of the save program. It first reads the control file containing the file offsets and re-builds the in-memory data structures from the binary data file. The first line in the control file contains the date and time of when it was created, followed by mission count and lines starting with either MH, ML or RQ meaning Mission Header, Mission Leg or scheduled ReQuest leg, respectively. After this, there is another count for unscheduled request legs. Following this count there may or may not be lines starting with the word RL, signifying the unscheduled request leg. The file format of the control file is as follows:

<Date and Time of file creation>

<mission count>

MH <mission_id> <file_offset> <mission_leg_count>

ML <mission_leg_id> <file_offset> <mission_leg_count>

RQ <request_leg_id> <file_offset> <request_header_id> <request_hdr_offset>

...

<unscheduled request leg count>

RL <request_leg_id> <file_offset> <request_header_id> <request_hdr_offset>

Following is a sample control file:

13 MAR 95 22 13	// This file is created on March 13 '95 at 22:13.
4	// There are totally 4 missions.
MH -1 368 8	// Mission 1 starts at 368th byte in binary file.
ML -20 572 1	// First leg of mission 1.
RQ 8 728 3 2080	// Scheduled request leg.

ML -21 3236 2

RQ 8 728 3 2080

RQ 9 3392 3 2080

ML -22 4744 2

RQ 8 728 3 2080

RQ 9 3392 3 2080

ML -23 4900 0

ML -25 5212 1

RQ 10 5368 3 2080

ML -26 6720 0

ML -27 6876 1

RQ 12 7032 3 2080

MH -2 8384 2 // Mission 2

ML -28 8588 1

RQ 5 8744 2 10096b

ML -29 11252 0

MH -3 11408 2 // Mission 3

ML -30 11612 1

RQ 6 11768 2 10096

ML -31 13120 1

RQ 7 13276 2 10096

MH -4 14628 7 // Mission 4

ML -32 14832 1

RQ 1 14988 1 16340

ML -33 17496 2

RQ 1 14988 1 16340

RQ 2 17652 1 16340

ML -34 19004 2

RQ 1 14988 1 16340

RQ 2 17652 1 16340

ML -35 19160 2

RQ 1 14988 1 16340

RQ 2 17652 1 16340

ML -36 19316 0

ML -37 19472 0

ML -38 19628 0

1 // There is one unscheduled request leg.

RL 3 19988 1 16340 // Unscheduled request leg.

Screen interface for selecting a file for saving or loading is shown in Figure B.1. The file to be selected can either be typed in the Selection field or selected using mouse from Files list. Once the file is selected press OK for further processing. Filter button is used to filter the list of files shown in Files list. In the sample screen Filter field shows "/project/phase2/usafe/shami_db/db_usafe/soln/*", a '*' at the end indicate list all files.

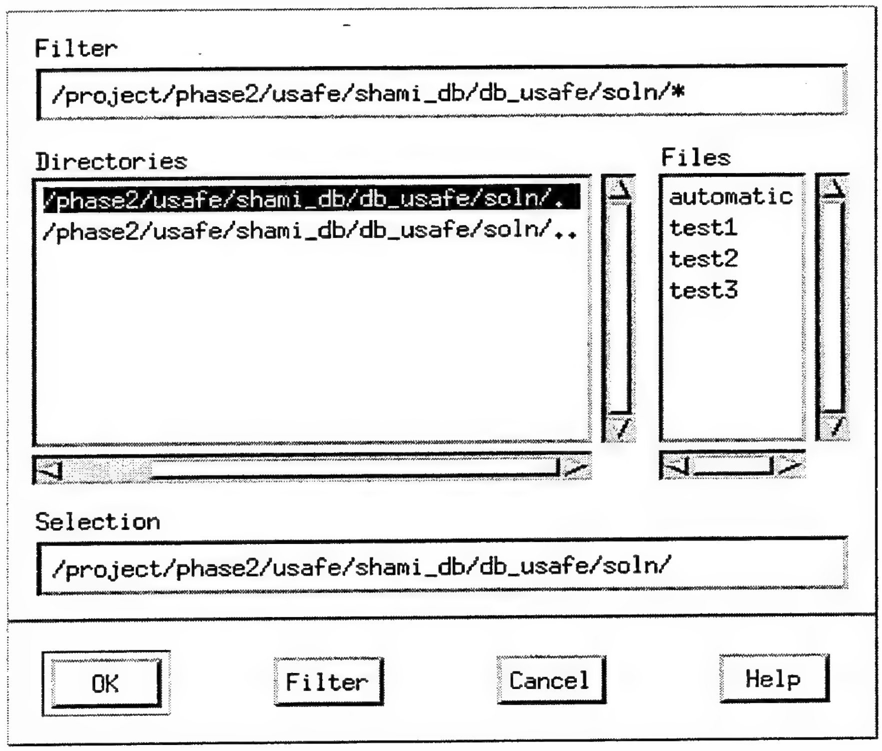

Figure B.1 File selection dialog box.

APPENDIX C-HYPERHELP SAMPLE FILES

The following is the sample hypertext input file. This file is parsed by the help compiler and loaded in the help database. The help database is later read by the run-time help module, whenever the user request for it. A sample help screen for this sample input file is in Figure C.1. A document can be navigated using indexes or history, dialog boxes for indexed navigation is in Figure C.2 and the dialog box for history-based traversal is in Figure C.3. Clicking on an item in the list will refresh the help screen with related help topic from help database.

```
<Comment hello this is a sample comment stmt>



<!DefineFont font1

 <Family lucidatypewriter>

 <pts 12pt>

 <Bold>

>



<!DefineFont font2

 <Family courier>

 <pts 14pt>

 <Bold>

>



<!DefinePara Heading

 <font3>

 <FirstIndent 75pt>

 <LeftIndent 75pt>

 <RightIndent 5pt>

>
```

<!DefinePara Section1

<font2>

<inter_lgap 5pt>

<FirstIndent 10pt>

<LeftIndent 20pt>

<RightIndent 30pt>

>

<!DefinePara Section2

<font1>

<inter_lgap 5pt>

<FirstIndent 40pt>

<LeftIndent 10pt>

<RightIndent 10pt>

>

<Heading intro INTRODUCTION TO USAFE SCHEDULING SYSTEM>

INTRO TO USAFE SCHEDULING SYSTEM

<Section1>

ISS is the Intelligent Scheduling System developed for USAFE by a research team at North dakota state univ. The application was developed in X window environment. There are totally 7 screens, they are <Hyper replacement mainwin MainWindow>, <Hyper replacement schd_win Scheduling window>, <Hyper replacement req_win Request entry window>, Aircraft entry window, Atlas window, Passenger window and finally candidate solution viewing window. In the subsequent sections we will briefly discuss about each of these windows.

<Heading mainwin Application Main Window>

Application Main Window

<Section1>

This is the main application window. From this window one could go to all other windows in the system using the button provided in the pulldown menu.

<Heading schd_win Scheduling Window>

Scheduling Window

<Section1>

Schedulers uses this window to schedule end user request to mission. These mission are created manually.

<Heading req_win Request Entry Window>

Request Entry Window

<Section1>

End-user request for flying from one place to another is entered into computer using this request entry screen. It validates all the data entered by the user before storing in the database.

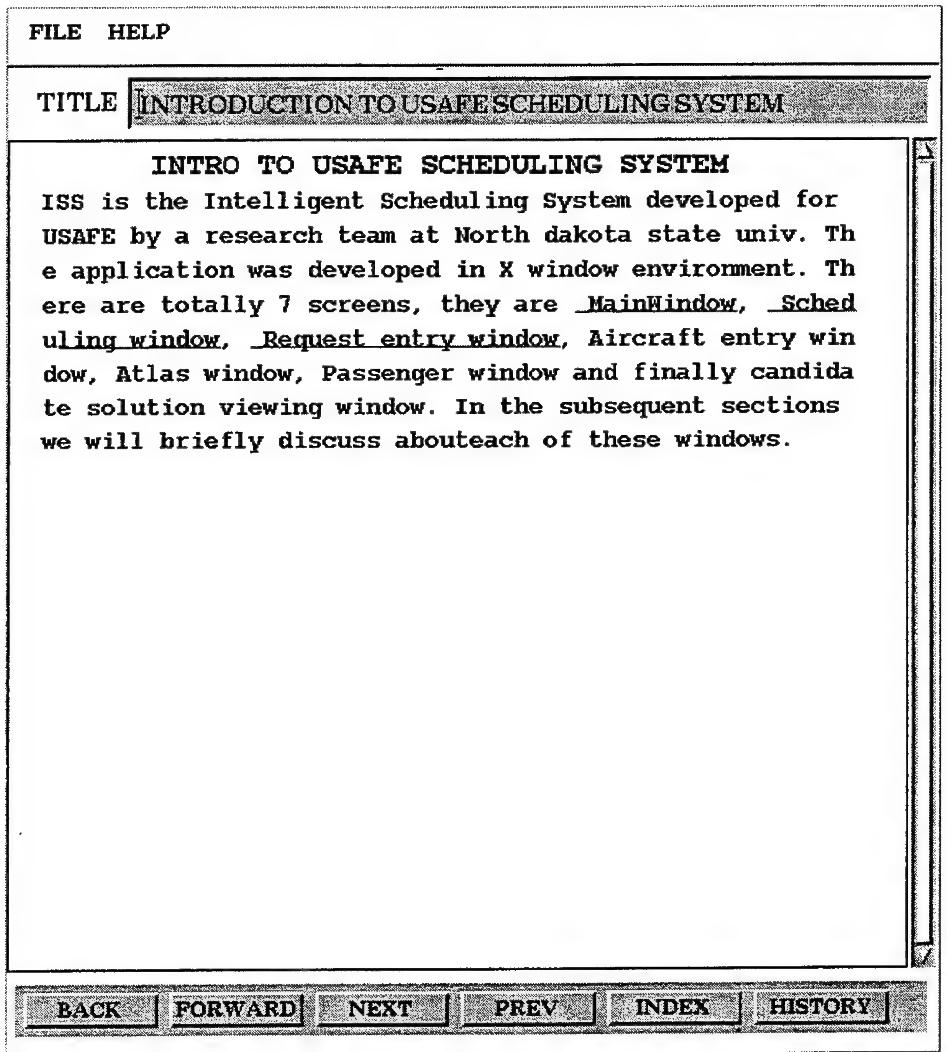

Figure C.1 Hypertext help screen.

Hot spots (hypertext) are underlined. Clicking on a hot spot using the mouse will replace the current help topic with the new selected help topic.

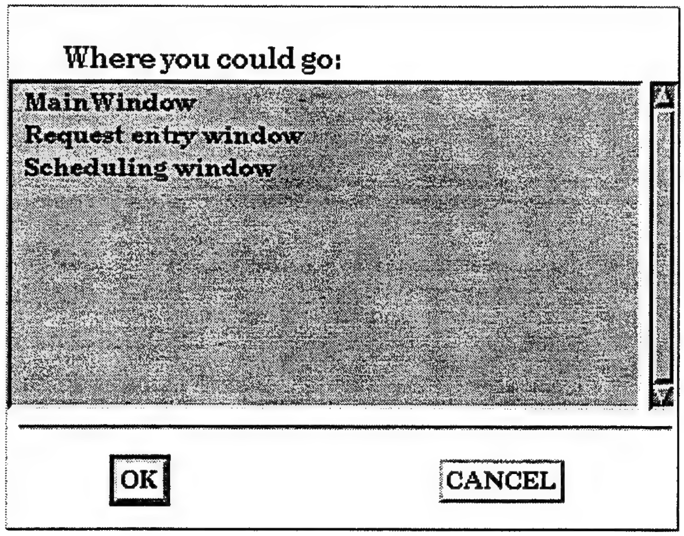

Figure C.2 Index Selection Dialog Box.

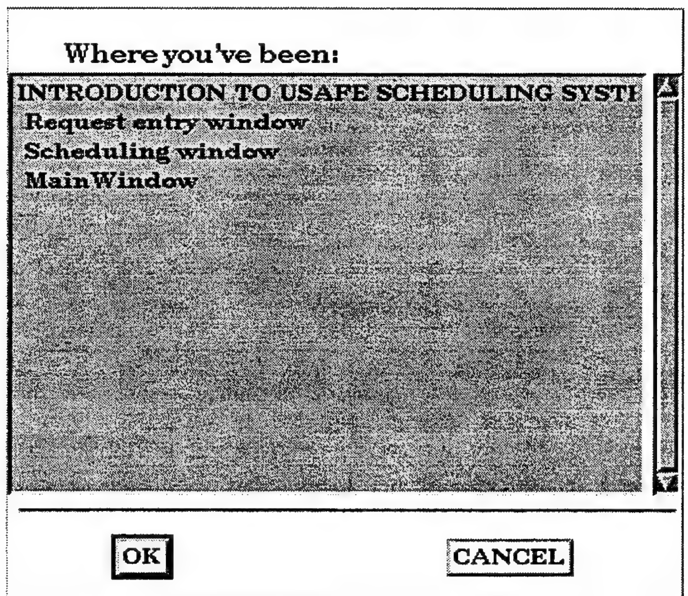

Figure C.3 History Selection Dialog Box.

APPENDIX D-UIL TO C SAMPLE FILES

Following is a sample .uil file.

```
module ModuleName

object

  Application : XmForm

    {

    arguments

      {

      XmNx = 37 ;

      XmNy = 17 ;

      XmNwidth = 820 ;

      XmNheight = 648 ;

      XmNborderWidth = 1 ;

      } ;

    controls

      {

      XmSeparatorGadget Separator1 ;

      XmLabelGadget LabelProblem ;

      } ;

    } ;

  LabelProblem: XmLabelGadget

    {

    arguments

      {

      XmNlabelString = compound_string("Problem:") ;

      XmNleftAttachment =                XmATTACH_FORM ;
```

```
        XmNtopAttachment = XmATTACH_FORM ;
        XmNleftOffset = 4 ;
        XmNtopOffset = 5 ;
        XmNwidth = 84 ;
        XmNheight = 17 ;
        } ;
    } ;
  Separator1: XmSeparatorGadget
    {
    arguments
        {
        XmNleftAttachment = XmATTACH_FORM ;
        XmNtopAttachment = XmATTACH_FORM ;
        XmNrightAttachment = XmATTACH_FORM ;
        XmNleftOffset = 0 ;
        XmNrightOffset = 0 ;
        XmNtopOffset = 25 ;
        } ;
    } ;

end module ;
```

C source file generated from above sample UIL file is:

```
/**********************************************************
 *  Following will be the C file generated from the above .uil code.
```

```
*************************************************************/

/*
 * WidgetHierarchy WidgetResource data structures are pertenent to
 * screen creation routines. It is found in resdef.h .
 */
#include <Xm/Form.h>
#include <Xm/LabelG.h>
#include <Xm/SeparatoG.h>
#include <resdef.h>

WidgetHierarchy ApplicationHierarchy[] = {
   { "toplevel",
     0, 0, 0, 0, 0, 0
   },
   { "Application",
    "toplevel",
    0,
    0,
    XM,
    XmCreateFormDialog,
    0
   },
   { "Separator1",
    "Application",
    0,
```

```
        0,
        XT,
        XtCreateManagedWidget,
        &xmSeparatorGadgetClass
    },
    { "LabelProblem",
        "Application",
        0,
        0,
        XT,
        XtCreateManagedWidget,
        &xmSeparatorGadgetClass
    }
};

WidgetReesources ApplicationResources[] = {
    {
        2, "Application", 5,
        XmNx, 37 ,
        XmNy, 17 ,
        XmNwidth, 820 ,
        XmNheight, 648 ,
        XmNborderWidth, 1
    } ,
    {
        2, "LabelProblem", 7,
```

```
        XmNleftAttachment, XmATTACH_FORM ,
        XmNtopAttachment, XmATTACH_FORM ,
        XmNleftOffset, 4 ,
        XmNtopOffset, 5 ,
        XmNwidth, 84 ,
        XmNheight, 17
    },
    {
        2, "Separator1", 6,
        XmNleftAttachment, XmATTACH_FORM ,
        XmNtopAttachment, XmATTACH_FORM ,
        XmNrightAttachment, XmATTACH_FORM ,
        XmNleftOffset, 0 ,
        XmNrightOffset, 0 ,
        XmNtopOffset, 25
    }
};

void
ApplicationCreation()
{

  int hiecnt;

  hiecnt = GetWidgetHierarchyIdx( ApplicationHierarchy, "LabelProblem" );
  XtVaSetValues( ApplicationResource[hiecnt].widget,
```